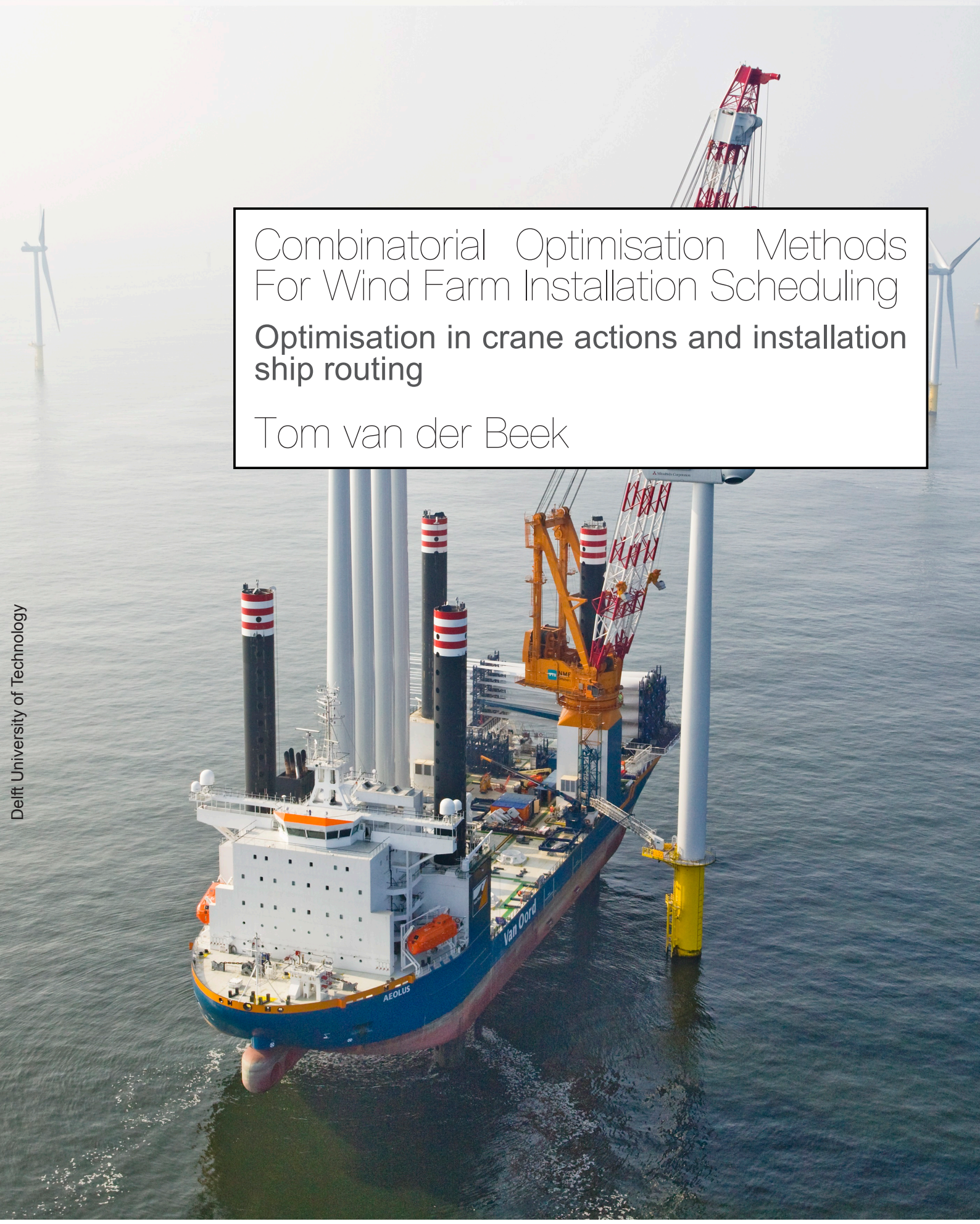
Combinatorial Optimisation Methods For Wind Farm Installation Scheduling

Optimisation in crane actions and installation ship routing

Tom van der Beek



Delft University of Technology

TU Delft Delft University of Technology
Deltares Enabling Delta Life
Van Oord Marine ingenuity

Combinatorial Optimisation Methods For Wind Farm Installation Scheduling

Optimisation in crane actions and installation ship routing
Master Thesis

Offshore and Dredging Engineering
Delft University of Technology
Graduation Date 15 June 2018

By

Tom Van Der Beek
4104145

Thesis Commission and supervision

Dr. ir. S.A. Miedema	Delft University of Technology
Dr. ir. J.T. Van Essen	Delft University of Technology

Thesis Commission

Dr. ir. J.F.J Pruijn	Delft University of Technology

Supervision

Prof. dr. ir. Karen Aardal	Delft University of Technology
Ir. Thijs Damsma	Van Oord
Dr. ir. Fedor Baart	Deltares

Acknowledgements

This thesis was a collaboration of two departments at TU Delft, namely Offshore And Dredging Engineering and the Optimization group of Applied mathematics. For me, my thesis period started almost a year before my real thesis, when i decided to explore different fields of interest. In this period i had numerous talks with different people at TU Delft about my wish to do a combined thesis. I am thankful for everyone who made time, and every talk helped me in making a better decision.

Furthermore, I would like to thank Dion Gijswijt and Ton van den Boom for sharing their views on my thesis. I would also like to thank Martina Fischetti, who I never met but was happy to generate an optimised wind farm layout for me. I also had multiple support meetings with Jeroen Pruijn, which were very valuable and I highly appreciated his effort and time.

I am happy i could do my thesis at Van Oord, and would like to thank Thijs Damsma from van Oord and Fedor Baart from Deltares for providing supervision. Before my thesis started, i already spoke a few times with Sape Miedema about my wishes. I would like to thank him, not only for supervision during my thesis, but also for understanding that doing something you enjoy is important.

Furthermore, i would especially like to thank my main supervisor from the Optimization department Theresia van Essen. When i decided that i wanted to learn more about optimisation, Theresia first help me with additional study in this field, then helped me to find an assignment and during that assignment provided me with great supervision. Without this, none of what i did in the last year would be possible.

When Theresia had to leave due to pregnancy, Karen Aardal substituted for her. Due to her, this absence had no impact on my thesis and I am thankful for all meetings and help.

Finally, i would like to thank my girlfriend Anastasiia. You were very supportive, especially during the final days when i was basically living behind my computer.

I was pleasantly surprised to see how many people, of most who i had never met before, made time and put in effort to help me. I conclude my thesis knowing that i found something which i truly enjoy doing, and in which i can develop my career. I think this is something very valuable, and once again i want to thank everyone who helped me with this.

Abstract

The offshore wind industry is rapidly growing, and so is the competition in this market. Normally, offshore installation companies receive a subsidy for building offshore wind farms, but due to multiple companies bidding to install for lower and lower subsidies, the point where wind farms are installed without subsidy has been reached. Therefore, companies have to install wind farms as efficient as possible. Since logistics are a major contributor to costs during the installation of an offshore wind farm, Van Oord and Deltares are doing research methods to solve logistic and scheduling problems. In this thesis, research is done in algorithms to solve two typical planning problems which arise during the installation of offshore wind farms: The Component Relocation Problem (CRP) and the Ship Installation Routing Problem (SIRP).

In the CRP, the storage and movement of turbine components in the harbour is considered. The crane operator manages the storage yard to fulfil a schedule of arriving and departing components. Since access of a storage location by crane can be hindered due to surrounding components, placement and relocation of components such that the requested component is accessible at the requested time, is a challenge. This challenge is solved in the component relocation problem, while minimizing the amount of relocations carried out by the crane.

A Mixed Integer Linear Programming (MILP) model is developed for this problem, along with an improved formulation to increase the solver performance and solve small instances to optimality. For practical use during offshore installation projects, a rolling horizon heuristic algorithm is developed.

In the SIRP, an installation schedule with corresponding ship routes is created. To install a turbine, multiple components have to be transported to the required locations, where they will be installed by multiple ships. While creating an installation schedule, it is desired to both minim the travelling time for ships, as the time ships are waiting on other ships and objects. An MILP for this scheduling problem is given, which models this problem as a Vehicle Routing Problem, with extensions to account for installation by multiple vehicles and time synchronization. Furthermore, to solve larger problems, an Adaptive Iterative Simulated Annealing (AISA) algorithm is developed.

The heuristic methods were validated against the exact methods and for small instances the heuristic methods were able to replicate the optimal solutions. The heuristic algorithms provided decisions in less than 12 hours, a relatively short time in comparison to installation times for offshore wind turbines. The AISA was able to create a schedule which drastically decreased the synchronization conflicts in a real world installation project. Although the sailing distance increased in comparison with the original planning, it was improved compared to the executed route. Furthermore, for the same project, the crane routing optimisation led to a reduction of 55% based on non-essential crane actions.

Contents

1 Introduction

One of the mayor current world challenges is to find a clean and sustainable form of energy. It is well known that generating power by burning fossil fuel impacts the environment negatively and depletes a finite resource. To meet the growing global energy demand, clean solutions are needed. IEA [2017] predicts that two-thirds of global investments in power plants to 2040 consists of investments in renewable energy. This percentage is even higher in the European Union: 80% of the new power-capacity will be green, with wind energy becoming the leading source of electricity after 2030.

By placing wind turbines at sea, offshore wind energy makes use of the generally more powerful wind at sea. Mostly located in Europe, offshore wind farms are a rapidly rising source of renewable energy. Currently, the existing wind farms still rely on subsidy to be profitable. These farms were awarded to companies which demanded a subsidy from the government in their bids. However, due to the competitive market, these subsidies declined and in 2017 the first project without subsidy was tendered.

This shows the growth of the industry, but is also highlights its competitiveness. It is crucial for offshore installation companies to match these zero-subsidy bids and thus install wind-farms as cost-efficient as possible. This thesis was done in cooperation with Van Oord and Deltares, who both have been exploring data and algorithmic based decision making. Van Oord is a global maritime contractor with its roots in dredging, although they expanded to offshore wind, oil and gas. Deltares is an institute for applied research in the field of water and subsurface, focusing on smart solutions and innovations.

Ait-Alla et al. [2013] stated that logistics account for 15% of the total costs during the installation of an offshore wind farm. This means that the optimisation of logistical processes plays a vital role in competing in the offshore wind farm installation market. However, this optimisation is currently done manually. This is a time-consuming process which might result in suboptimal schedules. Furthermore, work has to be redone when circumstances change, with usually very little time available.

In this thesis, optimisation methods are explored for scheduling problems in offshore wind farm installation. The problems encountered during scheduling will be analysed and abstracted. Subsequently, algorithmic methods will be developed to solve these problems. This includes modelling a problem, developing a scheduling method, validating this method and then applying it on the original problem. In order to do this, the field of mathematical optimisation is explored for similar problems and corresponding solution approaches.

These solution approaches usually consist of either, or both, a heuristic approach and an exact approach. All scheduling methods fall in either one of these two categories, and it is important to understand the difference between both. An exact optimisation method finds an optimal solution to a given model. When this solution is found, a certificate of optimality is included with it, meaning that one has the guarantee that for a given model no better solution exists. Unfortunately, for multiple problems, the time needed to find this solution

grows exponentially with the size of the problem. This is partly due to the enormous amount of potential solutions.

An example can be given for sailing between wind turbines. If a single ship has to visit all turbines in a field with 87 wind turbines, this results in $87! \approx 2.1 \cdot 10^{132}$ possible routes. To put this number in perspective, the approximate age of the universe is $4.35 \cdot 10^{17}$ seconds.

For this reason, optimisation methods have been created which use more efficient ways than simply evaluating all possible solutions. These methods decrease the time needed significantly, but unfortunately there are still many problems for which even these methods take too long. If this is the case, heuristic methods are used. These methods do not always find an optimal solution, and even if they do, there is no way of knowing if it is optimal. However, they find solutions within reasonable time and therefore have a practical use.

These optimisation methods are explored for two problems. First, for the component relocation problem (CRP). During an installation project, a harbour near the installation site is used for storage. Different material is stored here, and in the CRP the storage of turbine components is evaluated. These turbine components are parts of the turbines and will be installed later.

When these components arrive in the harbour, they are stored until requested by the installation ships. To efficiently use the limited storage space, the crane operator has to manage the storage area to make sure the correct components can be retrieved when needed for installation. This managing includes relocating the components, and since these are usually over 25 m high, this is no trivial task. Considerable costs are associated with each relocation, both due to working hours and risk. The CRP tries to find a solution for managing the storage area by crane, while minimizing the amount of moves.

Furthermore, the Ship Installation Routing Problem (SIRP) is introduced. This problem considers an offshore wind turbine installation project with a given amount of ships and turbine components. These components have to be transported to the installation site, where they are installed with the use of multiple ships and/or other objects. The ship installation routing problem consists of defining the ship routes in the installation field, together with the order and method of installation.

Ship rental is very expensive, which is why it is important to make good scheduling decisions to minimise the rental times needed. Furthermore, it is expected [Lütjen and Karimi, 2012] that installation ships will become the bottleneck during installation projects. This means that extra time needed for these ships also delay other activities.

In this thesis, algorithms are developed with the goal of minimizing the costs due to both of these problems. For each problem, a heuristic method is given for practical use, and this heuristic method is validated by comparing it against an exact method. The structure of this thesis is as following: First, the problem will be described in Chapter 2. This includes a description of an offshore installation project and a more detailed evaluation of the challenges encountered.

After this, a real-world project is analysed in Chapter 3. This is done to explore the significance of the discussed problems.

Since this thesis is aimed at both the field of mathematics and offshore engineering, an introduction to combinatorial optimisation is given in Chapter 4. Combinatorial optimisation consists out of finding an optimal object from a finite set of objects. Both the crane as routing problems can be categorised under this. Chapter 4 is aimed at readers with no or limited knowledge in the field of combinatorial optimisation. After this, the literature review is given in Chapter 5. Here it is explored what is done both in offshore scheduling optimisation and in other fields of interest.

Next, the model for the component relocation problem is given in Chapter 6, along with an exact optimisation method in Chapter 7 and a heuristic in Chapter 8.

The same structure is used for the ship installation routing problem. The model is given in Chapter 9, the exact optimisation method in Chapter 10 and the heuristic in Chapter 11.

Finally, the methods are tested on both virtual test instances as real-world data. This is done in Chapter 12. Finally, this thesis is concluded with a discussion, recommendations and a conclusion.

2

Logistics during wind farm installation

2.1. Offshore wind farm installation

The installation of an offshore wind farm is a challenging operation. First, a design is made. This design includes, amongst others, the locations and specifications of each wind turbine. A wind turbine exists of multiple components. A ground-based turbine exists of a foundation, a transition piece and the turbine itself. The foundation is placed on the seabed, and rises to near the surface. Examples of different types of foundations are monopiles, jackets or tripods as shown in Figure 2.1.

A transition piece is placed on top of this foundation. Transition pieces provide operational functionalities such as boat landings, ladders and platforms for access to the turbine. It also provides the connection between foundation and turbine. The turbine is installed on top of the transition piece which results in a complete turbine as shown in Figure 2.3.

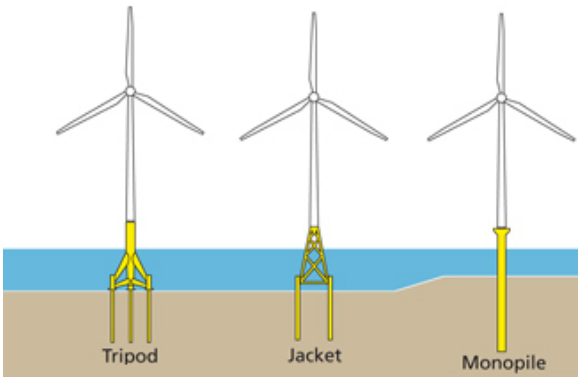


Figure 2.1: Wind turbine foundations

The design of the wind farm includes, amongst others, the locations of the turbines and the components used at each location. Due to environmental differences like water depth or soil properties, each component is unique. When all design specifications are decided upon, a production schedule is made in cooperation with the manufacturers. These manufacturers are usually located in different countries. Therefore, the components are shipped from the manufacturers to a harbour near the installation field.

Upon arrival, a crane is used to unload the components from the transportation ship to the loadout positions. Since there is limited unloading space, components have to be relocated from the loadout side, further into the storage yard. This is shown in Figure 2.2. When the components are requested for installation, they are relocated from the loadout side to the transportation ship and subsequently transported to the installation site for installation. As storage space is usually limited, the installation of turbines starts as soon as the first components arrive. The flow of components is shown in Figure 2.4.

Transportation to the installation site can be done either by an installation ship or by a separate transportation ship, depending on the project. Upon arrival, the components are installed. It is possible to first install a foundation and later return to install the transition

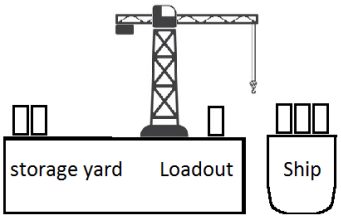


Figure 2.2: Flow of components from factories to the installation site

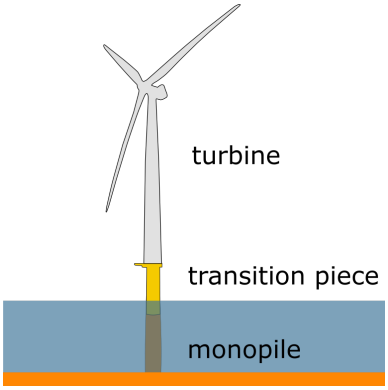


Figure 2.3: Wind turbine with foundation

piece, but it also happens that the same ship installs both directly after each other. The exact order of installation combined with the ship tasks for installation and transportation are defined in the ship routing schedule.

This describes the general logistics during the installation of an offshore wind farm. To analyse the real challenges, a more detailed description is needed. For this reason, a real wind farm installation project will be described in the next section. This allows for more insight in the challenges during the planning of logistics of an offshore wind farm installation project.

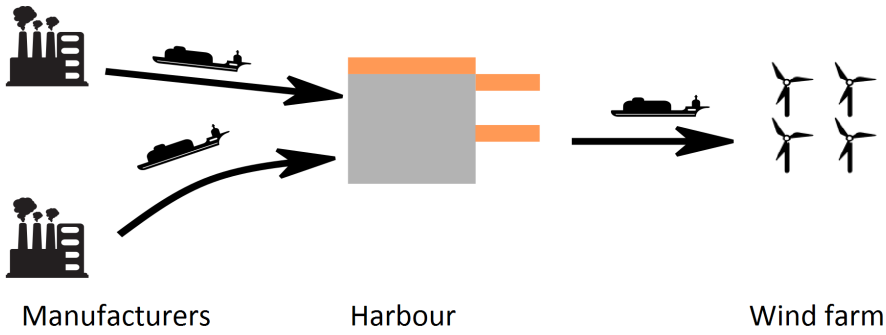


Figure 2.4: Flow of components from factories to the installation site

2.2. Walney Project description

The Walney offshore wind farm is located 14 km west of Walney Island in the Irish sea and consisted initially out of 102 turbines. In 2017, construction began on the Walney extension, where an additional 87 turbines were installed. Van Oord was responsible for installing the turbine foundations and transition pieces during this extension.

Figure 2.5: Svanen

In the Walney project, the foundations consisted of monopiles. A monopile (MP) is a relatively simple design consisting of a cylindrical steel tube. This tube is hammered partly into the seabed. On top of these monopiles, the transition pieces are installed to form the connection between the monopile and the wind turbine. The goal of the Walney project for Van Oord is thus to install all MPs and TPs, which can be seen in Section 2.2, at the specified locations.

Figure 2.6: Monopiles (left) and Transition Pieces (right)

2.2.1. Material

Two large installation ships are used during the Walney Project: The Svanen and the Aeolus, as shown respectively in Figure 2.5 and Figure 2.7. The Aeolus is a multi-purpose ship, which can both install and transport transition pieces and monopiles. It is even capable of doing this in one trip, meaning that it can install a complete turbine foundation without the need of other ships. The other installation ship, the Svanen, is less versatile. It can be described as a floating crane, being capable of only installation tasks.

Figure 2.7: Aeolus

In addition to these installation ships, tugboats can also be used to transport monopiles. For this to be possible, the monopiles have to be buoyant. This is done by using buoyancy plugs. Buoyancy plugs, in the rest of this thesis referred to as simply plugs, are large sealing disks, placed on both ends of the monopile, trapping the air inside. This causes the monopiles to stay afloat while being transported by tugboats.

2.2.2. Turbine foundation Installation

By using the Svanen and the Aeolus, two installation methods were available during the Walney Project. First there is the *Aeolus-only* method. In this method the Aeolus transports and installs a complete foundation independently. Secondly, there is the *combined* method, where the Svanen, Aeolus and a set of tugboats are needed for transportation and installation of a turbine.

When installing according to the Aeolus-only method, two monopiles and two transition pieces are picked up by the Aeolus in the harbour. These are subsequently transported to the field, where both will be installed by using the jack-up capabilities of the Aeolus. During jack

up, the vessel lowers four legs to the seabed to elevate itself, in order to increase stability. Although this allows the Aeolus to install complete foundations, it can only be used as long as the water depth is less than the length of the legs. For this reason, the Aeolus-only method is not possible for all turbine locations.

The other method uses tugboats to transport the monopiles to the installation field. Here, the tugboat will meet the Svanen at the turbine installation location. The monopile is transferred there to the Svanen and the buoyancy plugs are removed. After this, the tugboat is not needed anymore and can return to the harbour. The Svanen installs the monopile. Since the Svanen is an installation-only vessel, it does not leave the installation field and moves only between installation locations. The buoyancy plugs will be picked up by tugboats.

The second part of the *combined* method exists of the Aeolus transporting up to 5 transition pieces per trip, sailing to the field, and installing these on top of the earlier installed monopiles. This can be done without jack-up, thus the *combined* method can be used in all water depths. Installing without jack-up by the Aeolus is called floating installation. To switch between jack-up and floating installation, some modifications have to be done on the Aeolus. This takes around 10 days. It is important to finish installing all Aeolus-only turbines before starting with the Aeolus part of the combined method to avoid multiple modification periods.

2.3. Onshore Component Relocation

The Walney project is a good example of a complicated scheduling project. This term is used because it involves multiple interconnected logistical and scheduling problems. To create an optimal schedule, first the different challenges have to be evaluated. The onshore optimisation challenges are described in this section. These are related to the following question: How should the crane relocate components, in order to handle loading and unloading in the harbour efficiently?

After arriving at the harbour near the installation field, components will be stored, relocated and loaded onto the installation vessels by a crane. The movement of this crane can be limited by the stored components. Due to the large sizes involved, a relocation takes a considerable amount of working hours and includes the risk of damaging one or more components. If this would happen, the result would not only be costs of damaging the material, but also possible delay in the installation. It is therefore desired to manage the components with as few relocations as possible. In this thesis, the relocation of the transition pieces is considered. These are stored vertically on a grid of concrete foundation places.

2.3.1. Field

As stated before, the relocation of transition pieces is considered in this thesis. However, the goal is to find a solution method which is applicable to other objects as well. Therefore, the characteristics of a storage and relocation problem will be given and instead of transition pieces, components in general are discussed.

A typical storage area is shown in Figure 2.8. The locations can be divided into two categories, the loadout locations and the storage-only locations. Both parts consist of a grid where the components can be stored. Physically, the loadout area is located directly next to the water. Ships arrive here for loading and unloading, and the crane will then transfer TPs between the vessel and the loadout area.

From this loadout area, components can be relocated to the storage-only area. Here is more space available, but it is not possible to transfer directly between the storage field and the vessel. Moving a component to the storage-only field is optional: As long as there is enough space available, it is allowed for objects to stay in the loadout field.

2.3.2. Relocation constraints

The relocation constraints define whether relocation from or to a certain location is possible, depending on the layout of the storage area. These constraints are imposed due to physical

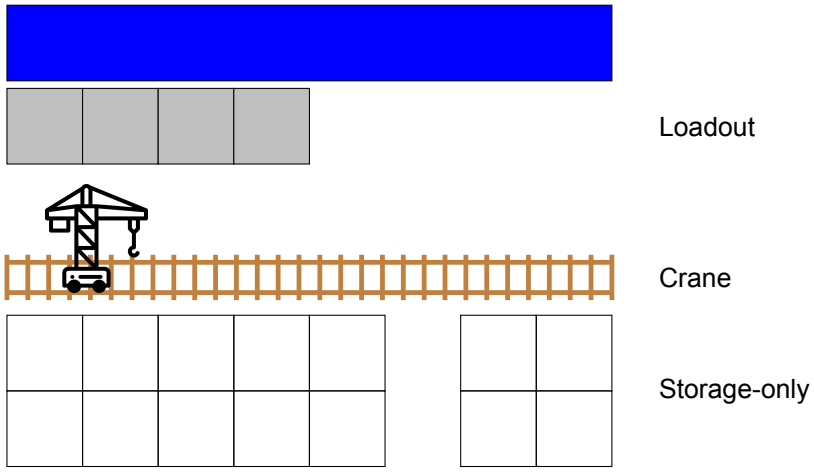


Figure 2.8: Storage field for TPs with the loadout area in orange and the storage-only area in green

properties of the storage field, crane and components. The CRP would be trivial without these constraints.

Because these constraints are results of physical properties, they are different per component. In this thesis, the relocation constraints are handled for transition pieces. These are relocated by a crane, which only can move over the crane track, as shown in Figure 2.8. A position is called blocked if a relocation from or to here not possible. There are two requirements for a position to be not blocked. First, there must be no component between the crane track and the position. In Figure 2.9, it can be seen that positions (2,3) and (2,6) are blocked due to this.

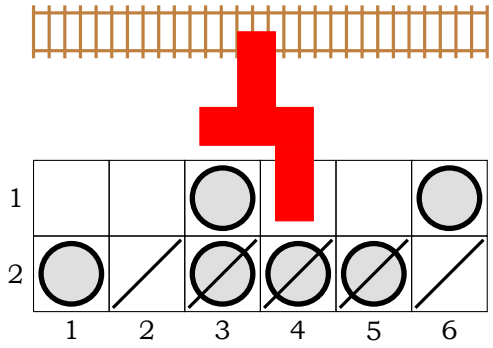


Figure 2.9: Relocation constraints

Besides this, there also should be an empty space left or right of the location. It can be seen that positions (2,2) and (2,4) are blocked because of this reason. If the location is not at the row directly next to the crane track, the position in front of the empty side neighbours must also be free. Position (2,5) has an empty side neighbour (2,6), but this side neighbour does not have an empty space between itself and the crane due to the TP at (1,6). For this reason, the TP at (2,5) is blocked.

These relocations constraints have been visualised in Figure 2.9 by the red crane. Although this does not even remotely resemble the real crane, this illustration can be used to quickly understand the relocation constraints. When the red part, or the horizontally mirrored version of this, is able to extract from the crane track to wrap around the desired location, that location can be seen as unblocked. These same constraints hold for the loadout area. The position of the loadout area relative to the crane track can be seen in Figure 2.8.

2.3.3. Phases

The crane operations can be divided in three phases: Unloading, pre-marshalling and loading. When components are delivered, the first phase is naturally unloading the components from the delivery ship to the loadout area. Due to the relocation constraints, it is important to place the TPs at this phase, based on where they will be stored later.

Between the loading and unloading phase, there is the pre-marshalling phase. The goal of this phase is to relocate the TPs such that during the loading and unloading phases, they can be transferred directly between loadout and ship. A pre-marshalling phase before an unloading phase will therefore include freeing enough space in the loadout area. A pre-

marshalling phase before a loading phase will place the correct TPs in the loadout area.

Besides these relocations, the pre-marshalling phase might also contain general storage management. It might very well be possible that before placing some TPs from or to the loadout area, other relocations can be done to minimise crane activities later in the project. This type of moves is also allowed in the pre-marshalling phase.

Finally, the loading phase consists of transferring the components from the loadout area onto the ship, after which they are transported to the installation field. This is a simple phase, because the order of loading for one shipload does not matter and the required TPs are already places on the loadout area during the pre-marshalling phase.

2.4. Offshore optimisation challenges

After storage in the harbour, the components are transported to the installation field, where they are installed. This creates different scheduling challenges as well, and in this section these challenges will be presented.

2.4.1. Installation methods

First, there is the choice of an installation method per turbine. As discussed above, some turbines can be installed only by the *combined* method and some can be installed by both methods. This is a hard constraint while deciding the installation method per turbine. However, this is not the only thing to consider. For the turbines which are compatible with both methods, a choice still needs to be made. The amount of turbines installed per method influences the duration of ship rental periods and deciding the specific method per turbine will affect ship availability and routing.

2.4.2. Ship routing

The methods and order in which the turbines will be installed have a direct effect on the sailing distance. The travelling time can be divided into two groups: Intrafield and outerfield travelling. The former refers to travelling between turbines in the installation field, and the latter refers to travelling between the installation field and the harbour. For the Aeolus, travelling between harbour and field is by far the largest contributor to sailing time. Unfortunately, there is not much optimisation possible by scheduling in this aspect. The only choice which impacts this is the amount of turbines installed by each method, since for the *Aeolus-only* method more trips are needed due to the lower transportation capacity.

However, even though sailing from and to the harbour takes up the most time for the Aeolus, intrafield traveling is not negligible. Savings in intrafield traveling will still be directly translated to savings in ship rental time. The opposite holds of course for an increase in intrafield travelling time. It is therefore important to consider sailing time while solving the other scheduling problems.

The opposite holds for the Svanen. Since it stays continuously in the installation field, intrafield travelling is the largest contributor to sailing distance for the Svanen. Exceptions are when the ship is relocating at the begin and end of the project, and when it should be relocated during bad weather.

Another aspect to consider is that for the combined installation method both the Svanen and the Aeolus are needed to install the turbine foundation. Of course the monopile has to be installed before the transition piece, so an important consideration during ship scheduling is making sure that the Aeolus only installs the TP after the Svanen installs the MP.

2.4.3. Buoyancy Plugs

Buoyancy plugs are used for transportation of the monopiles. These are fitted on both ends of the MP, thus sealing air inside. Since the monopiles can vary in size, a single plug can only fit on a subset of all monopiles. For each monopile, it is defined which plugs fit at the bottom and which plugs fit at the top. During the preparation of monopiles, installing the plug takes about 7 hours. After this, the plugs and MP will be inspected and tested for leakages, which takes another three hours. When reusing the plug, this results in a time gap of at least ten hours between arrival and departure.

These two conditions create another scheduling problem. If the installation of monopiles are scheduled in the wrong order, it might result in all plugs of a certain size being in the field or in preparation. If at this moment another plug is required of the same size, the transportation of a certain monopile to the installation field has to be delayed until the plug becomes available. Because of this, it is important to schedule the installation order such that waiting on plugs does not occur. This often influences sailing time.

2.4.4. Component availability

Turbine components are very large. Monopiles in the Walney project can have diameters up to 8 m and lengths of 80 m. Transition pieces have a height of 35 m and weight up to 400 000 kg. These components are usually fabricated across multiple countries. For the Walney project, the components were fabricated in Germany and Denmark.

From here, they are transported to the installation harbour where they are stored until they are requested for installation. The cost of this storage area depends both on storage size as duration. For this reason, it is cost-effective to start installing when the first components arrive, since this minimises both the required rental duration of the storage area and the maximum amount of components stored simultaneously. This means that while the project is carried out, components will arrive in batches at certain time intervals.

This restricts the freedom in the installation order. Before starting the project, it is possible to discuss with the manufacturer the installation order. Subsequently, a schedule will be made for the arrival of components. If changes are to be made to the schedule after this, it has to be according to the delivery order.

Although there is a fixed production schedule for the components, it is quite possible that a deviation occurs from this. Components might take longer to fabricate, or they might not pass the inspection and will have to be repaired. When this happens, a different component is shipped to allow the installation project to continue and to utilise all space available in the transportation ship.

Component availability thus has a large effect on the scheduling decisions. Firstly, it creates an obligation to broadly commit to the original schedule, since this schedule defines the schedule of the manufacturer. Secondly, it is possible that during the execution of the project deviations occur in the manufacturers schedule, which force deviations in the installation schedule.

2.5. Problem definition

In the current chapter, two scheduling problems were presented, the SIRP and the CRP. While finding a schedule for the SIRP, the main points to consider are time synchronization between ships and/or objects, intrafield travelling of ships and the choice of installation methods. For the CRP, the goal is to find a schedule for crane actions taking in to account the blocking constraints due to physical properties of the crane, storage area and components.

The scope of this thesis is defined as the decisions between arrival of the wind turbine components in the harbour up to the installation. The components arrive following a fixed schedule. After this, the following steps are decided by the optimisation: The movement of the component in the storage field, the way and time of transportation to the installation field, and the method of installation.

Before creating the solution methods, a model has to be created to define the problems exactly. The solution method for these problems has the goal of finding an optimal solution for the given model within the time available during the planning phase and within available time during the execution of the project. It will be shown later that no method has been found which does both, and therefore two algorithms are required per problem: One which will find the optimal solution for the given model, and one which gives a good solution within the available time.

The storage field consists of a grid based structure of locations where components can be stored. Relocations between these locations are done accordingly to the required in- and outflow of components and the spatial relocation constraints. These constraints define when

components can be moved, based on neighbouring components of both the source as the target location of a relocation move. The amount of crane movements is minimised in order to reduce costs.

From the storage area, the components are transported to the installation location. For each ship, it is defined if it can transport a certain component, and if so, how many per trip. It might also be possible that objects like buoyancy plugs are required during transportation. Therefore, decisions have to be made when to transport a certain component and in which way.

Upon arrival of components at the installation location, they have to be installed by a set of ships. One or more installation methods are defined for each turbine. Installation methods consist of a set of instructions which have to be carried out in order. Per instruction, it is defined which component is needed and which ship. When and by which ship these instructions are carried out is decided by the scheduling method.

To summarise, the scheduling method will define an order of crane movements and an order of ship tasks which include loading, travelling and installing. These schedules are then varied to minimise the cost of the project. Costs taken into consideration are ship rental costs and estimated costs per crane action. Based on this, the following research questions now can be defined:

1. Ship Installation Routing Problem
   - (i) Given a fixed set of ships, components, available actions and installation requirements, how can the global minimum project costs based on ship and harbour rental costs be reached by determining the ship routes and actions?
   - (ii) What algorithm can be used to find a, possibly non-optimal, solution for this problem within the time available during both pre-project planning and during project execution?
2. Component Movement Problem
   - (i) Given an arrival and departure schedule of components, the geometry of the storage location and the rules regarding allowed relocations based on neighbouring components, how can the crane movement schedule be found that results in a global minimum of crane movements?
   - (ii) What algorithm can be used to find a, possibly non-optimal, solution for this problem within the time available during both pre-project planning and during project execution?

In the next chapter, an analysis of the Walney project is given. Although the goal of this thesis is to find a method applicable to multiple projects, analysing a single one can give insights in the significance of certain choices, and to find out where there is room for improvement.

3

Analysis of Walney project

3.1. Planning

To analyse optimisation opportunities, the existing schedule of the Walney Installation project was evaluated. Two separate schedules were considered. First, the installation routing schedule consists of the routes and installation tasks for each ship. This thus defines when each component is transported to the installation field and by which method it is installed. Secondly, the crane schedule contains all required crane movements to manage the storage area.

The crane movement planning contains almost enough information to know exactly which component is placed where. Only in a very few cases, where the choice will not affect the outcome, some relocations were not documented. However, the installation routing schedule has more gaps. First of all, it is assumed that synchronization time, where ships have to wait on components or vice versa, does not occur. Furthermore, all intrafield travelling time is set to a constant time to relocate the ships between any two turbines. It is important to know what the effect is of these assumptions. Therefore, the routing schedule will be compared to the executed installation.

3.2. Installation choices

It was mentioned earlier that the choice of turbines per installation method is the most defining factor for the rental periods per ship. In Table 3.1, it can be seen that in both the scheduled as the executed planning, the majority of turbines are installed by the *combined* installation method.

	Scheduled	**Executed**
Combined	53	56
Aeolus-only	34	31

Table 3.1: Turbines installed per installation method

The number of choices per installation method in the executed planning are quite similar to the scheduled planning. In the end, only three turbine installation methods deviate from the original planning, all of which were scheduled by the *Aeolus-only* method and are switched to the *combined* installation method.

3.3. Intrafield sailing

While analysing the intrafield routes, two things are inspected. Firstly, the sailing distances and times are evaluated. This is interesting since it gives an indication of the significance of intrafield routing. Secondly, it is investigated how much the executed routing deviates from the original one. This is done to understand the required flexibility in planning and to evaluate the original planning's robustness.

The sailing distances and times are shown in Table 3.2. The total scheduled sailing time is calculated based on the average sailing speed in the execution. Two things stand out

	Average (km)	Total (km)	Time total (Hours)
Scheduled	1.8	246	87
Executed	2.8	323	120
Increase	56 %	32 %	38 %

Table 3.2: Intrafield sailing

when looking at Table 3.2: Firstly, the time and distance of intrafield sailing has increased significantly during the execution. Secondly, both ships combined spend in total 5 days on intrafield traveling. In Appendix A.3, it is estimated that the cost of these 5 days is around €530.000.

To get an estimate of the similarity in routes between planning and execution, the following simple metric has been created: For each ship visit to a turbine i in the planning, the neighbours are defined as the turbines which have been visited on the same trip, directly before or after turbine i. The score for this visit to turbine i is then the fraction of similar neighbours in the execution. The total similarity is then defined as the average over all visits. This can be expressed as shown in equation (3.1). For identical schedules, it would hold that $s_k = 1$.

Variable	Description
s_k	Similarity metric for ship k
T_k	All turbines visited by ship k
N_{is}^k	Neighbours of visit to turbine i by ship k in planning
N_{ie}^k	Neighbours of visit to turbine i by ship k in execution

Table 3.3: Variables for Equation (3.1)

$$s_k = \frac{1}{|T_k|} \sum_{i \in T_k} \frac{|N_{is}^k \cap N_{ie}^k|}{|N_{is}^k|} \tag{3.1}$$

This similarity metric is calculated for both ships and has a value of 0.66 and 0.37 for the Svanen and the Aeolus, respectively. This shows that there is a large difference between the planned and executed routing. This difference is the largest for the Aeolus. It can therefore be concluded that the real routes sailed deviate a lot from the routes in the original planning, and unfortunately, this deviation causes a very significant increase in sailing distance.

3.4. Waiting time

In the original planning, the assumption was made that as soon as the Svanen reaches its location, the monopile is already waiting there and can be installed directly. This assumption turns out to be invalid when looking at the executed project. In total, the Svanen has spent 96 hours, thus 4 days, waiting on location for the monopile to arrive. A quick calculation, included in Appendix A.3, estimates the cost of waiting at €301.000. It is also observed that waiting on monopiles happens quite regularly over the duration of the project. There is no clear recognizable downwards trend.

As explained earlier, monopiles can only be transported when plugs of the correct sizes are available in the harbour. Due to the limited amount of plugs it can occur that the requested plugs are not yet available and the transportation of the monopile is delayed. As shown, this has created a significant amount of delay during the execution of the Walney project.

3.5. Crane actions

The planned schedule for crane movements was evaluated as well. The storage plan uses the scheduled installation order and defines each TP relocation done. Three types of movements have been identified: (Un)loading, loadout preparation and reshuffling. (Un)loading moves are defined as the transfer between a vessel and the loadout area. Loadout preparation moves are moves where a component is positioned from the loadout area to the storage field after being unloaded, or vice versa before being loaded. Reshuffling contains all other moves.

Each TP has two loading moves, on which no optimisation is possible unless the harbour loading is dismissed entirely. Additionally, there are either two or zero loadout preparation moves. In most cases it will be two, but sometimes it might possible for a component to remain in the loadout place and not move to the storage area. Reshuffling can be seen as storage field management. There is no straightforward upper or lower bound for these moves.

In the planned movement schedule, it is obvious that the 87 TPs amount to 174 loading moves. As stated before, there is no apparent room for improvement here. The amount of loadout preparation is more interesting. Here, 158 moves are done. This means that 8 TPs resided solely in the loadout area, while the rest used the storage-only area. The amount of reshuffling moves is 31. This puts an upper bound for the moves to be reduced at 189. An estimate per location move was put on €1000 per crane move, so minimizing crane actions during projects might reduce costs significantly.

3.6. Discussion

During the analysis of this project, some assumptions have been made. Travel velocities were taken constant, based on distances divided by travel durations. This overestimates the times somewhat on larger distances since vessels will reach higher velocities here. Anchoring time was reported separately, which made it possible to exclude this time before calculating the velocities.

It also must be noted that although the data was analysed with care, the Walney project lacked a uniform, automated way of documenting. Excel files with nearly 5000 entries combined were created and evaluated manually, which gives room for small errors to slip in.

Additionally, there is an issue with the comparison of the schedule and execution. The sailing route of the Svanen was assumed continuous in both. In reality, there were certain breaks due to weather during the execution of the project. During these breaks the Svanen had to move away from the installation field to wait for better weather. To be able to compare the execution with the schedule, these breaks were not taken into account and it was assumed that the Svanen sailed directly between turbines. In reality, there is no reason that the turbine before and after the break should be near each other, however, in the comparison it would still be viewed as a bad solution if they are not.

3.7. Conclusion

Multiple conclusions can be drawn from this analysis of the Walney. First, it can be seen that both intrafield travelling as components relocations by crane account for significant costs during the installation of the Walney Wind Farm. Furthermore, the comparison between planning and execution reveals the deviation in the latter one. This deviation causes an increase in the travelling time. A possible explanation is that there is a lot of time available for the initial planning, but during the execution, decisions have to be made in a much shorter time period. This might result in changes needed to the initial schedule that are not optimal.

Additionally, it was shown in the analysis that the Svanen spends a large amount of time waiting on monopiles. This waiting time was not documented in the initial planning. This waiting also persisted during the whole project, suggesting that no effective actions were taken to mitigate this waiting time.

For the crane movements, only the planned schedule was evaluated. Due to the nature of this problem, it is difficult to see if there is room for improvement, but an analysis of the

planned crane movements made it clear that there is a significant cost caused by moving the TPs between locations in the harbour, and that it is worthwhile investigating if this can be improved.

Because of these reasons, it can be concluded that improving the planning methodology based on intrafield travelling and crane movements might result in significant reduction in project costs. This new methodology needs to fulfil the following requirements: First, it has to take interconnected problems into consideration. In the original planning the travelled distance was done quite efficiently, but plug preparation was not considered enough and therefore delay occurred. Secondly, the planning needs to handle the dynamic nature of offshore projects. Before starting the project there is abundant time to plan and schedule, but as soon as the project start circumstances will change and the planning has to be adjusted. At that moment, there is much less time available. An improved planning methodology should be able to handle these changes with scarce planning time.

A method of creating a planning can be put into the form of a finite series of instructions. Such a series is called an algorithm. When an algorithm is implemented in a programming language it is called a computer program. In this thesis, it will be explored how to create an algorithm to solve both scheduling problems presented in this chapter.

Creating an optimal schedule can be abstracted as finding an optimal object, the schedule, from a finite set of objects, namely all possible schedules. The research field which consists of techniques of solving this problem is called combinatorial optimisation.

In the next chapter, a basic introduction to this field is given.

4

Combinatorial optimisation

4.1. Introduction

Both the CRP and the SIRP can be seen as problems in combinatorial optimisation, which is a field in mathematical optimisation. It studies optimisation problems with discrete solutions. In this chapter, an introduction to this topic is given. The goal of this is to provide a general overview of combinatorial optimisation, and a more detailed description of theory relevant to this thesis. First, some information is given on the technique of modelling problems. After this, some problems of relevance to this project along with some problem-theory will be presented. The final part of this chapter discusses some techniques for solving problems. Here, general techniques are included as well as some very problem-specific methods, if relevant to this thesis.

4.2. Formulation

To solve an optimisation problem, first a mathematical model has to be formulated. This includes the essential features of the real world problem. Like with most modelling, there is a trade-off between the level of detail and performance. Adding more details will make the model more similar to the real situation, but it will quite often also increase the difficulty of solving it. It is therefore important to understand which features are significant and how the solving procedure will change by adding them.

A general optimisation problem consists of a cost function $f(x)$ and a set of feasible solutions $\mathcal{F}$. The goal is to find a solution from this set which minimises (or maximises) the cost function $f(x)$. For a minimization problem, this can be notated as following:

$$\text{Find } \hat{x} \in \mathcal{F}, \text{ such that } f(\hat{x}) \leq f(x) \forall x \in \mathcal{F}.$$

or as:

$$(P) = \begin{cases} \min \quad f(x), \\ \text{subject to} \quad x \in \mathcal{F}, \end{cases} \tag{4.1}$$

4.2.1. Linear Programming

An important property of these models is whether they are linear or not. In a linear model, the set $\mathcal{F}$ is defined by constraints, and both the constraints and cost function consist of a linear combination of the decision variables. These problems can be solved with linear programming (LP). A linear optimisation problem can be expressed in the canonical form:

$$f(x) = c_1 x_1 + \ldots + c_n x_n = c^T x \tag{4.2a}$$

subject to

$$Ax \leq b \tag{4.2b}$$

with

$$x, c \in \mathbb{R}^n, A \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^d \tag{4.2c}$$

Here n is the amount of decision variables and d is the amount of constraints. Equation (4.2c) specifies that x and c are vectors of dimension n, b a vector of dimension d and A a matrix of dimension $n \times d$.

Every constraint in $Ax \leq b$ forms a hyperplane which cuts the solution space. The resulting set of solutions is a polyhedron in n-dimensional space. A bounded polyhedron is called a polytope. This is illustrated in Figure 4.1 for $n = 2$. The corners of this polytope are called vertices. An important property of a linear optimisation program, is that if there is an optimal solution, at least one of these is located on a vertex of the polytope.

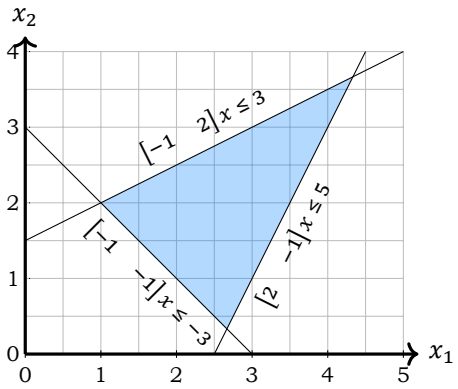


Figure 4.1: Polytope of $Ax \leq b$

4.2.2. Duality

An important property of an LP is duality. Consider the primal problem (P):

$$(P) = \begin{cases} \min & c^\intercal x \\ \text{subject to} & Ax \geq b \\ & x \geq 0 \end{cases} \tag{4.3}$$

A dual problem (D) can then be defined as:

$$(D) = \begin{cases} \max & b^\intercal y \\ \text{subject to} & A^\intercal x \leq c \\ & y \geq 0 \end{cases} \tag{4.4}$$

The variables x And y are feasible solutions for (P) and (D) respectively. From duality theory, it then follows that $c^\intercal x \geq b^\intercal y$. Let x^* and y^* be the optimal solutions for (P) and (D) respectively, then the strong duality theorem [Cook et al., 1998] states that $c^\intercal x^* = b^\intercal y^*$. These properties are very useful, since they can be used to verify if a solution is optimal, or find an optimal solution for one problem if it is known for the other. By using the duality theorem, it is possible to gather information about the existence of optima based on the existence of feasible solutions for the primal or dual problem.

4.2.3. Mixed Integer Linear Programming

A mixed integer linear programming (MILP) model is an LP model where certain variables can only take on integer values. In practice, a lot of problems include binary variables representing yes or no. This can be modelled by using variables which can take on 0 and 1.

An example can be given for the shortest path problem. This problem consists of a network of nodes with paths between them as seen in Figure 4.2. The starting node is node S and the ending node is node T. For each pair of nodes, the distance c_{ij} is defined as the distance

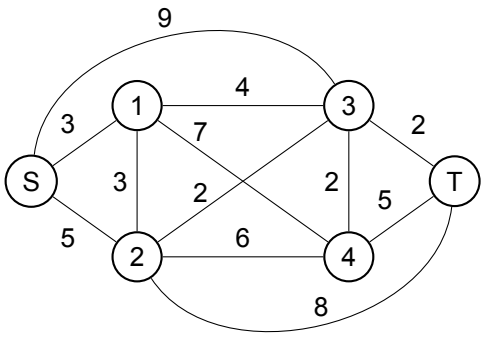


Figure 4.2: Example of shortest path network

between node i and node j. The mathematical model then can be described as:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.5a}$$

subject to

$$\sum_{\{j|(S,j)\in A\}} x_{Sj} = 1 \tag{4.5b}$$

$$\sum_{\{i|(i,T)\in A\}} x_{iT} = 1 \tag{4.5c}$$

$$\sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{k|(k,i)\in A\}} x_{ki} = 0 \forall i \in V \setminus \{S,T\} \tag{4.5d}$$

$$x_{ij} \in \{0,1\} \forall (i,j) \in A \tag{4.5e}$$

(4.5f)

Variable	Description
x_{ij}	If equal to 1, the path from node i to j is in the shortest path. If equal to 0, it is not.
A	Set of all arcs (paths) in the network
V	Set of all nodes in the network
S	Starting node
T	Ending node
c_{ij}	Cost of going from node i to j

Table 4.1: Variables for Equations 4.5

The cost function is defined in Equation (4.5a) as the sum of the cost per connection, only if that connection is in the solution x. Equation (4.5b) counts all paths leaving from the starting node and sets it to 1. This means that in the solution exactly one path is leaving from the starting node. In Equation (4.5c), the same is done for the amount of connections travelling to the ending node. Equation (4.5d) specifies the path. For each node, except the ending and starting node, it sets the connections going in equal to that of the connections going out. This means that if there is a path travelling to a node, there will also be a path leaving that node for each node except the start and end nodes. Finally, Equation (4.5e) defines that x consists of binary variables.

4.3. Problems

In optimisation theory there is often a similarity between multiple problems. By evaluating a certain problem and looking with which problem it has similarities, solution methods can be found. Quite often, a new problem is a variant or a combination of very well studied problems. For this reason, some standard problems are discussed here.

4.3.1. Travelling Salesman Problem

A classic and one of the most studied problems in optimisation theory is the Travelling Salesman Problem (TSP). In this problem there is a network of roads and cities and a salesman at a starting position. The goal is to find the shortest route possible, while visiting each city and then returning to the starting position. An example of a solution to a real-world TSP can be seen in Figure 4.3.

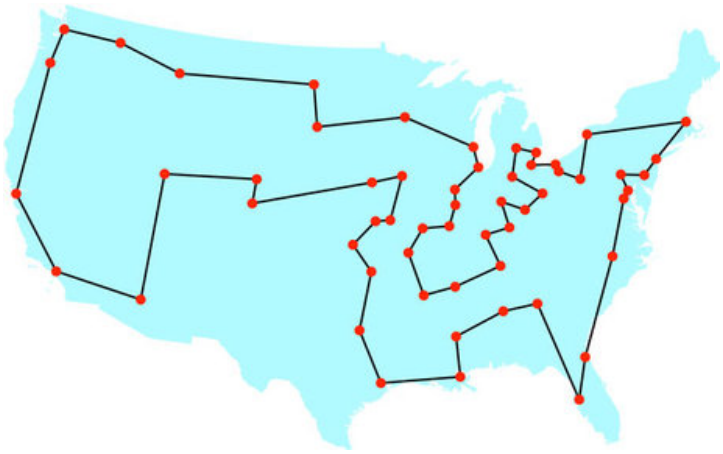


Figure 4.3: Solution to real world TSP

This basic problem has a large number of applications. These are in logistics, but for example also in the manufacturing of microchips or routing of electricity cables. A mayor difficulty in solving the TSP lays in preventing subtours. This can be seen in the MILP formulation given in Equations 4.6. Here, A is the set of all arcs in the network and N the set of all nodes. The decision variable x_{ij} is one if arc (i, j) is traversed in the solution, and zero otherwise. To form a connected route, Equations (4.6b) and (4.6c) are introduced. These define that for each node, there is exactly one incoming arc and one outgoing arc. It might seem that these constraints define a connected tour, but unfortunately this is not the case.

$$\text{Min} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.6a}$$

subject to

$$\sum_{i\in N} x_{ij} = 1 \forall j \in N \tag{4.6b}$$

$$\sum_{j\in N} x_{ij} = 1 \forall i \in N \tag{4.6c}$$

$$\sum_{i\in S, j\in S} x_{ij} \leq |S| - 1 \forall S \subset N, |S| > 0 \tag{4.6d}$$

$$x_{ij} \in \{0, 1\} \forall i \in N, j \in N \tag{4.6e}$$

The reason for this is illustrated in Figure 4.4. Here, two solutions to a 4-node TSP are shown. On the left-side a correct solution is shown, while a wrong solution is shown on the

right-side. It is obvious that this solution is wrong, since it is not a single tour. However, this solution does satisfy Equations (4.6b) and (4.6c), since each node has exactly one incoming arc and one outgoing arc. This shows the main problem in solving the TSP: a solution with multiple subtours also satisfies Equations (4.6b) and (4.6c).

Figure 4.4: Left: Correct solution. Right: Solution with subtours

To prevent these solutions with subtours, Equation (4.6d) is introduced. This constraint adds one inequality for each subset of nodes $S \subset N$. It specifies that two times the selected arcs between nodes in S is equal to the size of S minus one. Since each node has two connected arcs, and the sum of all arcs connected to nodes in S is equal to two times the arcs between nodes in S plus all nodes from outside S to S, Equation (4.7) holds. Combining this with Equation (4.6d) gives Equation (4.8), which states that for each set of nodes S, except for the empty set or for $S = N$, there must be at least two arcs entering or leaving S. This eliminates the subtours. However, the amount of subsets of N is equal to $2^{|N|}$. This means that the amount of constraints grows exponentially, quickly resulting in an excessive amount of constraints. Therefore, the MILP as shown in Equations 4.6 can usually not be entered directly in an MILP solver. Later in this thesis, approaches will be presented to overcome this problem.

$$2 \sum_{i \in S, j \in S} x_{ij} + \delta S = 2|S| \forall S \subset N, |S| > 0 \tag{4.7}$$

$$\delta S \geq 2 \forall S \subset N, |S| > 0 \tag{4.8}$$

4.3.2. Vehicle Routing Problem

The vehicle routing problem (VRP) is a generalization of the TSP, introduced by Dantzig and Ramser [1959]. The problem was extended to support multiple travellers, now called vehicles. Each of these vehicles has to make a route to visit a number of nodes and then return to the starting position, and combined all nodes have to be visited.

In contrast to the TSP, in the original VRP each vehicle has a capacity and each node has a demand. The total sum of the demand of all nodes visited by a single vehicle cannot exceed its capacity. While satisfying this constraint, the VRP as formulated by Dantzig minimises to total distance travelled.

Note that the above statements were made regarding to the original VRP. Since this formulation was introduced, many variations were created. For example, time windows can be introduced in which the node have to be visited, or the restriction is added that certain nodes can only be visited by certain vehicles. The amount of variations is vast, and more research on this will be presented in Chapter 5.

4.3.3. Flow Shop Problem

Another classic problem is the Flow Shop Problem (FSP). This is a special case of Job Shop scheduling. Job Shop scheduling is an optimisation problem, where there is an amount of jobs which has to be scheduled to machines for certain operations. For each job at any time at most one operation is processed, and each machine at any time is also processing at most one operation. An example solution can be seen in Figure 4.5.

The FSP is a Job Shop problem which includes is a given sequence for the operations per job, meaning that for any job the operations have to be in a certain order. This can be seen as a production process, for example. Each item needs to pass certain machines, but in a set order. The FSP solver will then search for a schedule with the lowest total time.

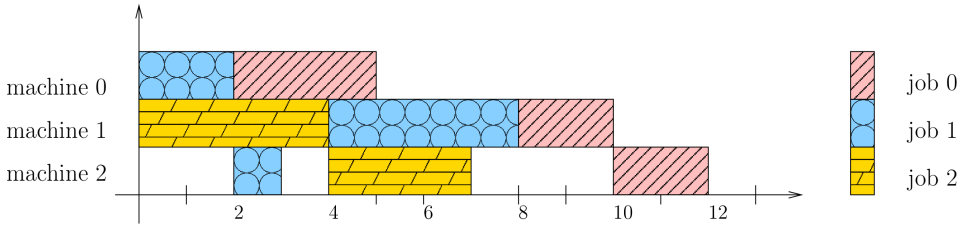


Figure 4.5: A solution to a job shop problem

4.4. Problem Complexity

When a solution method for a certain problem is found, it can be evaluated on efficiency. Problems are classified by the amount of resources it takes to solve them and to verify a solution. Algorithms are called efficient when they run in polynomial time in the input size of the problem. This means that the time it takes to solve a problem scales by a polynomial function of the problem size. This is notated as $O(\ldots)$. For example, an algorithm which optimises the relocation of n cars on r roads, can have an order of $O(n^2 r)$. This means that the time has a quadratic relationship with the amount of cars and a linear relationship with the amount of roads.

Algorithms which scale faster than polynomial will quickly become very large. This can be illustrated with an example. Let's say, a computer is able to solve a certain problem for $n = 10.000$. A new computer is bought, with twice the speed of the original one.

If the problem would scale with $O(n)$, the new computer can now solve problems for $n = 20.000$, a large improvement. If the algorithm would be $O(n^2)$, the problem can now be solved to $n = 14.142$. This of course is still a large improvement. However, if the algorithm would scale with $O(2^n)$, the capacity would increase by 1, thus the new computer would be able to solve problems up to $n = 10.001$. If the algorithm would be $O(n!)$, the new computer should be 10.001 times as fast as the old one, on order to have an increase of $n = 1$. This shows that for certain algorithms, increasing computational power has almost no effect.

Because of this distinction in scaling, problems will be classified in complexity-classes. First, let us define what a problem is. There are two types of problems. The first type asks you to find a certain object. For example, find a path from a to b. The other type is a decision problem: a problem which can be answered with yes or no. An example of this can be: Check if path P is a path between nodes a and b.

Optimisation problems fall in the first category, since in these problems an object has to be found with an optimal cost according to some cost function. Every optimisation problem has a corresponding decision problem. This can be shown for the shortest path problem. The optimisation problem is then: Find a shortest path between nodes a and b. The corresponding decision problem would be: Given a number r, does there exist a path between nodes a and b with length of at most r?

The class of problems which can be solved in polynomial time is called $\mathcal{P}$. Problems in this class will also be referred to as easy problems. As stated before, each optimisation problem has a corresponding decision problem. If for every instance that this decision problem has a positive answer, there exists a certificate for which the correctness can be verified in polynomial time, the problem is in the class $\mathcal{NP}$, which stands for non-deterministic polynomial. This certificate proves the answer. For example, if the question would be: Is there a path between nodes a and b with a lower cost than 20? Then a path between a and b with a cost lower than 20 would be a certificate for that problem. It is not necessary that the certificate can be found in polynomial time, only that it can be verified.

It follows that all problems in $\mathcal{P}$ are also in $\mathcal{NP}$. If the reverse would be true, it would mean that $\mathcal{P} = \mathcal{NP}$ and that an algorithm can be found to efficiently solve every problem in $\mathcal{NP}$. This is however not known, as it is one of the big open questions in computer science.

Another subclass of $\mathcal{NP}$ is the class $\mathcal{NP}$-complete. To explain this class, the term reduction is introduced. A reduction transforms one problem into another. When a problem L can

be reduced efficiently to problem H, it means that an algorithm which solves H efficiently can also solve L, and therefore L cannot be harder to solve than H. When every problem $L \in \mathcal{NP}$ can be reduced to a problem H in polynomial time, it means that H is at least as hard to solve as any problem in $\mathcal{NP}$. Problem H is then called $\mathcal{NP}$-hard. When a problem is both $\mathcal{NP}$-hard and in the class $\mathcal{NP}$, it is called $\mathcal{NP}$-complete. It follows that if an efficient solution for a problem $H \in \mathcal{NP}$-complete would be found, an efficient solution for every problem $L \in \mathcal{NP}$ is found and that $\mathcal{P}$ would equal $\mathcal{NP}$.

Problem complexity is an important part of combinatorial optimisation, since it shows what can be expected as the performance of the exact solution, and if heuristics are needed.

4.5. Finding Solutions

When a problem is properly formulated in constraint equations and a cost function, a method needs to be selected to solve this problem. An important property of a solution method is if it can give a guaranteed optimal solution. This means that it will provide an optimal solution and a guarantee that there is no other feasible solution with a better result. Due to the large solution spaces of some problems, finding a guaranteed optimum might not be possible in a reasonably amount of time. This holds especially for problems not shown to be in $\mathcal{P}$. Note that this does not mean that they are not in $\mathcal{P}$ because it is not known if $\mathcal{P} = \mathcal{NP}$, but for now it means that there is no efficient algorithm for these problems. For a lot of problems, methods can be found to find good solutions within a much smaller time. First, methods to find a guaranteed global optimum are discussed, followed by methods to find good but possibly sub-optimal solutions in less time. These solution methods which do not guarantee optimality are called heuristics.

4.5.1. Exact

Simplex

The simplex method is a method to quickly solve a linear optimisation problem. It was shown earlier that the solution space of an LP-problem forms a polytope P, as shown in Figure 4.1. If there is a solution, at least one optimal solution will be located at the corner points of P, called vertices. It also holds that when a vertex x is selected, there is a vertex reachable by crossing an edge of the polytope which has a better solution, if and only if x does not hold the optimal solution.

Therefore, the simplex can select any vertex x, and calculate the values at the neighbouring vertices. If none of these represent a better solution, x represents the optimal solution. If there is a better neighbouring solution x', the algorithm will replace x by x' and repeat the process until it reaches the optimal solution.

Branch and Bound

The simplex method makes it possible to find a solution to an LP problem in a very small amount of time. A problem arises when certain variables will change from continuous to integer. The problem is now called an MILP problem. For these kinds of problems, the simplex method is not usable anymore and the difficulty increases greatly. One method of solving an MILP model is the Branch and Bound (BB) method.

This algorithm consists of a exploring tree consisting of nodes and branches. Each node represents an LP-problem. The top node represents the initial MILP, without the integral requirement. Removing the integral requirement is called integer relaxation. The simplex-method is used to get an optimal solution x' for this problem. If x' is not integer for all required integer variables, branching occurs. Branching divides the problem in two separate problems by selecting a non-integer variable with value x_i', and creating two new problems by adding the constraints $x_i \leq \lfloor x_i' \rfloor$ and $x_i \geq \lceil x_i' \rceil$. Since x_i has to be integer, this move does not remove any feasible solutions.

This branching creates two new nodes, each representing a new LP problem. At each node, the optimal solution for the problem with integer relaxation represents a lower bound for all solutions created by further branching, since adding new constraints will never result

in a better optimum. This branching happens until a node is found with an integer solution. This solution can then be used to prune nodes with a lower bound higher than this value.

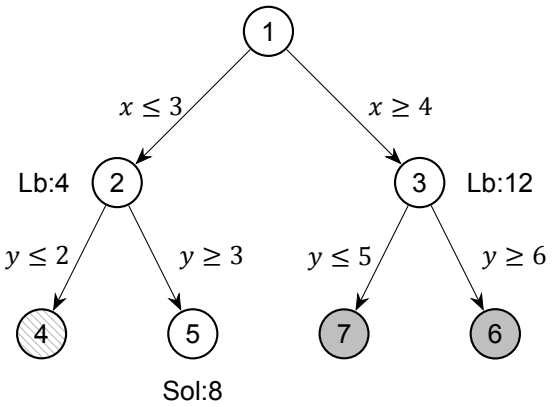


Figure 4.6: Branch and bound for two variables. Node 4 still has to be explored

An illustration of this algorithm is given in Figure 4.6. Here, the white nodes are the explored nodes. It can be seen that when node 5 is explored, a solution with the value of 8 is found. For node 3, the lower bound was found to be 12, so node 6 and 7 can never be better than the found solution and can be pruned. Node 4 still has to be explored. This shows how the algorithm can only explore a part of all solutions, but still can guarantee optimality. It is also possible that the relaxed LP does not give a solution. If this is the case, it means that none of the nodes in the rest of the branch will contain feasible solutions, and thus those branches can also be pruned.

It is desired to have the lower bound as high as possible. This will result in more branches being cut off, and therefore guaranteeing optimality without having to explore every node in the tree. Another important method to decrease computational time is symmetry breaking. Symmetry can be defined as different solutions which hold the same results. This can be illustrated by considering a problem where a postman has to plan a route to deliver multiple packages and then return home, in a symmetric network. In a symmetric network, each path between two nodes has the same length as the reversed path. When an optimal solution is found, a symmetric version of this solution is the same path, but reversed. This symmetric solution represents nodes in the branch and bound tree and since it does not provide new information, it might as well be made infeasible. If a constraint is added to remove these solutions, less branches have to be explored which will result in less computational burden.

Branch and Cut

The Branch and Bound method is a great way to solve MILP problems when the constraints and variables can be defined up front. We encounter however that sometimes there are too many variables or constraints. When there are too many constraints, the Branch and Cut (BC) algorithm is used. This algorithm generates cuts (constraints) during the execution. The principle of BC is to select a subset of cuts which fully restrict the solution space. Figure 4.7 shows a flowchart of a BC algorithm.

It can be seen that most of the algorithm is similar to branch and bound. After solving the linear relaxation, BB has three options. If the value of the solution is larger than the best found integer solution, or no feasible solution is found, the node is pruned. If the value is smaller, the node is either branched if the solution is not integer or saved as new best solution when integer.

A BC algorithm starts with a subset of all constraints $\mathcal{L} \subset \mathcal{L}_n$. The LP is solved with this limited set of constraints $\mathcal{L}$. When a feasible solution x is found with a better value than the best solution so far x^* is found, a step is added in comparison with the BB algorithm. The algorithms checks if there is a constraint e in $\mathcal{L}_n$ and not in $\mathcal{L}$, which is violated by x. If this constraint is found, it is added to $\mathcal{L}$ and the linear relaxation of the current problem is

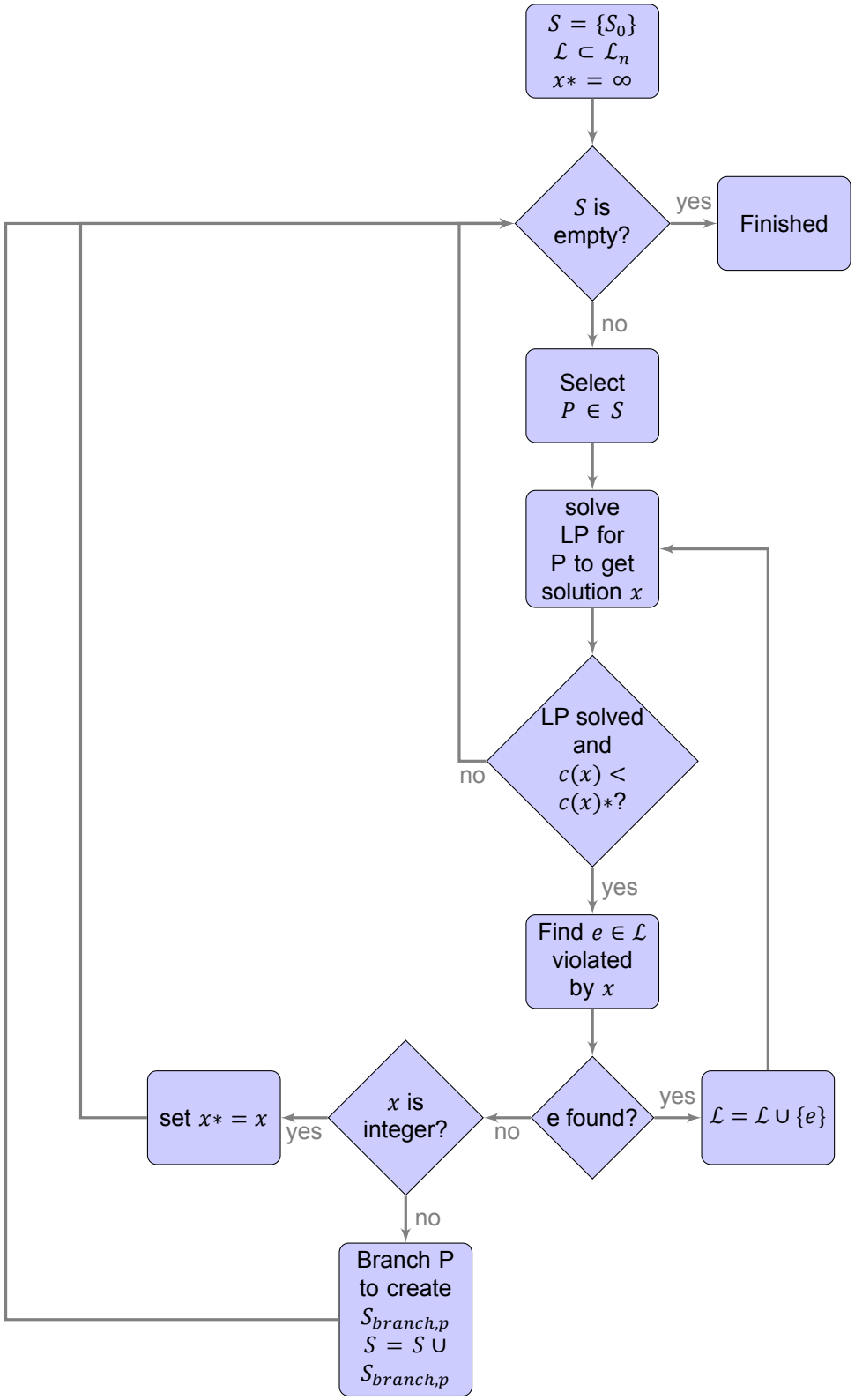


Figure 4.7: Branch and cut algorithm for a minimization problem

solved again until there is no $e \in \mathcal{L}_n \setminus \mathcal{L}$ violated by x. After this, the BC algorithm continues branching or saving the optimal solution like a BB algorithm. Let H describe the polytope of $\mathcal{L}$. The constraint e then separates x from H. This is the reason why the problem of finding e is also called the separation problem.

Branch and Price

In other problems, the amount of variables might be too large to initialise before branching. A Branch and Price (BP) algorithm is a variation on BB with variable generation. A variable corresponds to a column of the A matrix in equation (4.2b), so variable generation is also called column generation. The main idea of column generation is to find a subset of all variables, such that it can be guaranteed that adding other variables will not improve the solution. In this way, the optimal solution can be found without including every variable.

A detailed explanation of a BP algorithm was given in Feillet [2010] and was used, together with Desrochers et al. [1992] and Vance [1998] to derive the flowchart in Figure 4.8.

The Branch and Price algorithm works as following: The problem starts with an unbranched problem S_0 in the problem-pool S and with an initial set of columns R_1. With this, the restricted master problem $MP(R_1)$ is created. This is the master problem with only a subset of all possible columns. The integer relaxation is solved for this problem and the algorithm checks if there are any possible columns which will improve the solution. These are called columns with reduced cost. If these are found, they are added to R_1 and $MP(R_1)$ is solved again until no columns with reduced cost exist. After this, the regular branching procedures happens.

When the Restricted Master Problem is solved, it is compared to the dual of the complete Master Problem for violated constraints. A violated dual constraint corresponds to a column with reduced cost in the master problem [Feillet, 2010]. This can be used to find new variables which might improve the solution, and when no constraints are violated it guarantees that no variables with reduced cost can be found.

Cutting Planes

Besides formulating the problem, constraints can also be used to improve solver efficiency. This is called the cutting-plane method. Constraints will be added to decrease the solution space, without removing the optimal solution from it. An example can be seen in Figure 4.9. Here, the original constraints form the feasible region. The feasible solutions are the integer values within this region. By adding the cutting-plane constraint, the feasible region becomes smaller without removing any integer points, and therefore without removing the optimal solution.

There are also cases where feasible points can be removed. In these cases, there has to be a guarantee that the feasible points do not contain the only optimal solution. This can be done by for example symmetry breaking, as introduced earlier.

Dynamic Programming

Another way of solving to optimality is by dividing the problem into smaller (overlapping) subproblems. Solutions for these subproblems are stored and will be used to compute solutions for expanding problems. This is called dynamic programming.

This method is much more problem-specific than the branch and bound method. Take for example the traveling salesman problem. This problem includes a salesman in a network which has to visit multiple cities. For the subproblem of visiting n cities, the solution can be found be taking the solution for $n-1$ cities and adding the nth city at the right location. By starting at a subset of just one city and each time adding one, the optimal solution can be found.

Problem specific algorithms

For certain problems, specific algorithms have been created to solve them to optimality. If a problem can be formulated as one of these problems, the same algorithm can be used. An

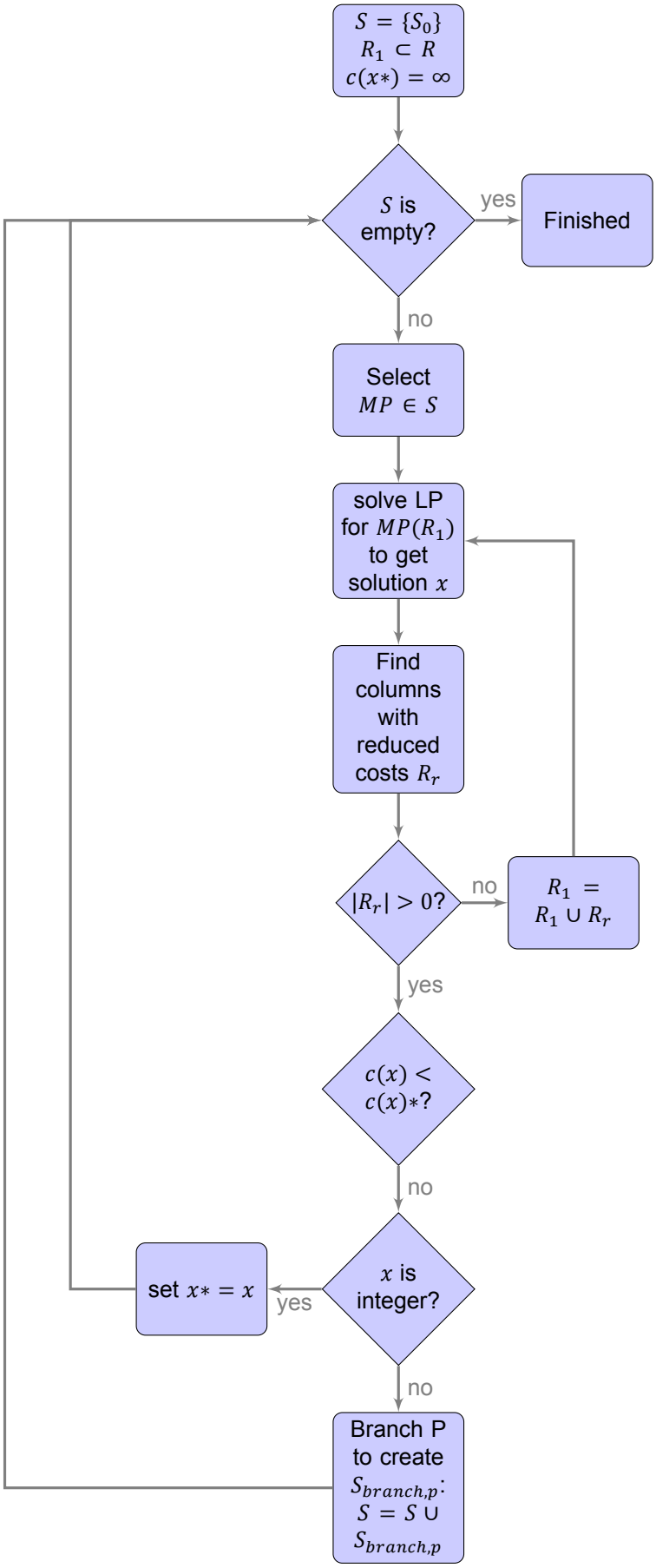


Figure 4.8: Branch and price algorithm

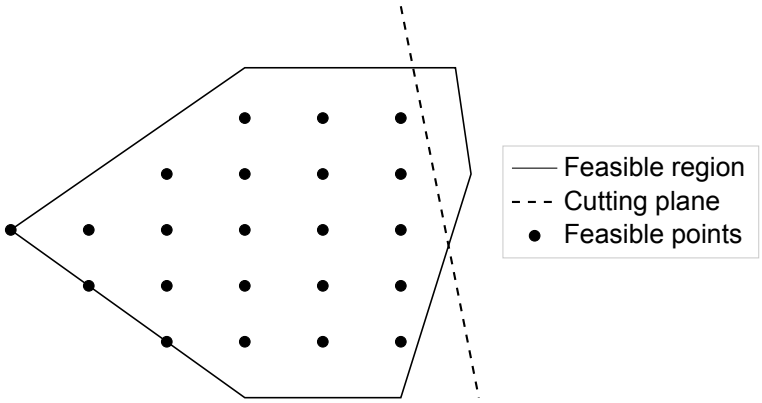


Figure 4.9: Cutting plane

example is the shortest path-problem, where a shortest path between two nodes has to be found in a network. Multiple algorithms have been created to solve this, for example Dijkstra [1959] or Hart et al. [1968].

For example, a problem with relocation containers can be modelled as a network by having each possible configuration as a vertex and an arc between each two vertices if one component can be moved from one configuration to the other. This is illustrated in Figure 4.10. It then follows that if a shortest path is found in this graph, a minimum number of moves is found to go from one configuration to another one. In the figure, the shortest path between two configurations is shown.

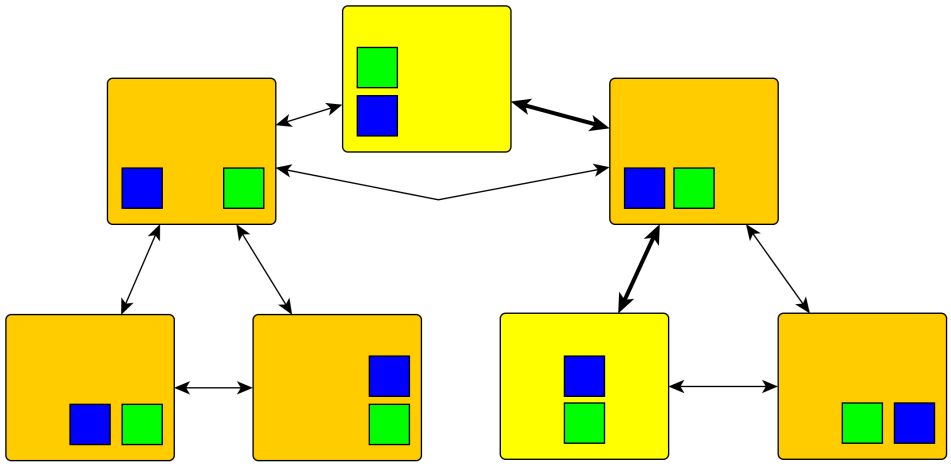


Figure 4.10: Network representation of relocating containers

4.5.2. Heuristics

Finding an optimal solution might not always be possible within the available timespan. In this case, heuristics are used. These are methods to find solutions which are not necessarily at an optimum, but are a lot faster than methods like branch and bound or dynamic programming.

Heuristics and optimal methods can help each other. Using a good heuristic can speed up the branch and bound method, since everything which has a worse lower bound than the heuristic solution can be cut off. On the other hand, optimal methods can be used to validate heuristics.

Meta-heuristics

Meta-heuristics are problem-independent techniques to create heuristics. Although, in literature there is not always a clear distinction between meta-heuristics and heuristics, the following definitions were found in Sörensen and Glover [2013] and will be used during this thesis:

> A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimisation algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimisation algorithm according to the guidelines expressed in such a framework
>
> In general, for a given problem, a heuristic procedure is a collection of rules or steps that guide one to a solution that may or may not be the best (optimal) solution.

Thus, a heuristic is problem-specific, and a meta-heuristic is a problem-independent method. For a problem specific implementation of a meta-heuristic, both terms can be used.

Grasp

Many meta-heuristics are based on improving an existing solution. This means that before starting the improvement algorithm, an initial solution has to be available. This is quite often done with a deterministic algorithm, but this would mean that each time the algorithm is ran, it will come to the same deterministic outcome. To explore a larger part of the solution space it is useful to run the LS multiple times with different initial solutions. One way to create these solutions is GRASP. GRASP stands for Greedy Randomised Adaptive Search Procedure and is found to be used in articles in relevant literature, as will be shown later. It was introduced by Feo and Resende [1995] and it is a heuristic procedure which iteratively creates a solution and optimises this with a LS method. The best solution is stored and returned at the end of the iteration.

Although introduced as a method for construction and further optimisation, GRASP's main contribution is the construction phase. In this phase, a feasible solution is iteratively constructed, one element at the time. A greedy function is used to order all candidate items at an iteration step. The function is a measure for the benefit of adding a certain item to the solution. It is adaptive since this function is updated with each iteration. From the generated list, an item is chosen randomly among the top candidates. This randomization step allows different solutions to be constructed each time the construction phase is ran. Algorithm 1 shows the construction phase of grasp.

Algorithm 1 Construction phase of GRASP

```
Solution = {}
repeat
    rcl = create_restricted_candidate_list()
    s = select_random(rcl)
    solution = solution ∪ s
    adapt_greedy_function()
until Solution complete
```

Local Search

Local search is an algorithm which explores the solution space to improve a feasible solution. The algorithm works as follows: It starts with an initial solution. One or more methods are defined to transform this solution into another feasible solution. The set of other solutions which can be found by applying on of these methods to the current solution is called the neighbourhood. In the context of routing, the neighbourhood might be to swap two nodes in the route. This is show in Figure 4.11. The neighbourhood of a solution then consists of all possible routes which can be found by swapping two nodes.

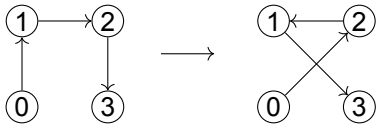


Figure 4.11: Example of swap move in local search for routing

Local search computes the solution in the neighbourhood and then moves to the best one. This is repeated until no improvement can be found any more, or a certain time or iteration limit is reached. An example is shown in Figure 4.12. Here the solution space is illustrated as a set of nodes where each node is a feasible solution. An arrow between nodes means that one solution can be reached from another node. The current solution is in black. The neighbourhood, in gray, is then the set of nodes to which an arrow is connected. If the neighbour with the best value has a better value than the current solution, that solution is set as the current solution and the process is repeated. This is shown in Algorithm 2 [Oh et al., 2014].

Algorithm 2 Local search algorithm

```
s = create_initial_solution()
repeat
    N_s = get_neighbourhood_solutions(s)
    s_n = select_best(N_s)
    if f(s_n) < f(s) then                    ▷Compare values
        s = s_n
    else:
        Break
    end if
until stopping criterion met
```

Neighbourhoods are problem dependant and are critical to the performance of a local search algorithm. But even with a well chosen neighbourhood, there is no guarantee that a global optimum will be found. A local optimum in a LS algorithm occurs when each solution in the neighbourhood has a worse value. At this moment, the algorithm terminates, but there still might be solutions outside the neighbourhood with better values. A lot of effort has been done in finding different algorithms which can escape these local optima.

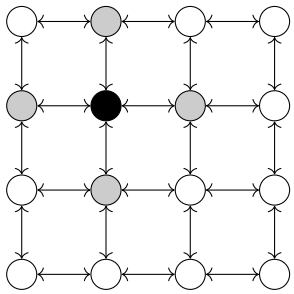


Figure 4.12: Local Search solution space. Each node is a feasible solution. The current solution is black, the neighbourhood of this solution is grey

Tabu search

One way to avoid getting stuck in local optima is the tabu search meta-heuristic. This is a LS variation introduced by Glover [1989]. Unlike the LS, it also accept worsening moves if no improving move is available. Although this prevents a full stop, it might create a cycle. Figure 4.13 illustrates this. When at position 2, it will move to the best neighbour: position 3. From there the best neighbour is 2, which starts a cycle between 2 and 3.

To mitigate cycling, a tabu list is kept of characteristics of the last n visited solutions. The algorithm will then only consider neighbours not in the tabu list. If the tabu list is long enough, it will prevent cycling long enough to escape local optima.

Simulated Annealing

Simulated annealing is a probabilistic local search, inspired by metallurgy [Kirkpatrick et al., 1983]. A temperature is introduced, which starts high and slowly decreases during the execution of the algorithm. At each iteration, a random neighbour is selected. If this neighbour

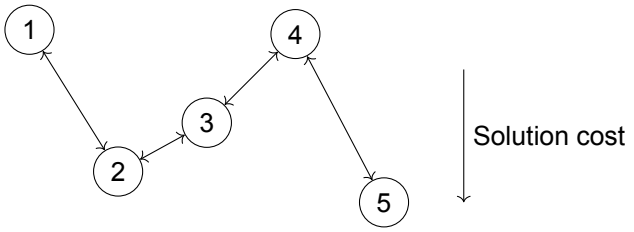


Figure 4.13: Tabu search illustration: When starting at 1,2 or 3,minimizing will create a cycle between nodes 2 and 3

has a better objective function value, the algorithm moves to that neighbour. If not, it moves to the neighbour with a probability based on the neighbour solution value and the temperature. When the temperature is high, it means that worsening solutions are easily accepted. As the temperature drops, the probability of accepting worse solutions decreases to zero.

Genetic algorithm

Local search methods can be represented by a single explorer walking through the solution space. Population based methods include multiple travellers. A genetic algorithm is a population based method inspired by evolution theory. Like evolution, it is based upon three main principles: Selection, mutation and reproduction.

The algorithm starts with an initial population of different solutions. For each solution, the cost is evaluated and a selection is made of individuals which will reproduce. For the reproduction step, multiple solutions will be combined to create one or more new solutions. Each solution, consisting of a combination of its parents, will also have a mutation. A mutation is a small random change, for example by switching some visits in VRP context.

After this process, a new generation is created, hopefully with better solution values. This process is then iterated until some stopping criterion is met.

Adaptive Large Neighbourhood Search

One technique which is used by authors of similar problems to the SIRP, as will be shown later in the literature survey, is Adaptive Large Neighbourhood Search (ALNS). This is a modified version of Large Neighbourhood Search (LNS). LNS is a local search variation and is based upon a process of continual relaxation and re-optimisation. Across literature, these same moves are also named destroy and repair, since the relaxation move destroys a current solution and the re-optimizing move repairs it. These two steps transform one solution to another and define the neighbourhood of a solution.

This means that instead of calculating multiple neighbourhood solutions and selecting the best one, a new solution is created and immediately validated. The destroy and repair algorithms usually contain some randomization to provide different solutions when no improvement is found the first time. Another modification to the classic LS is that there is a separate acceptance criterion to define if a solution should be accepted as the new initial solution. The solution with the best value so far is stored separately. This is shown in algorithm 3.

The ALNS was introduced by Ropke and Pisinger [2006] and a more general form was given in Pisinger and Ropke [2007]. The ALNS is an extension to the LNS which chooses adaptively among a number of relaxation and re-optimisation techniques. Like the LNS, the ALNS can be based on any local search method, e.g. tabu-search, simulated annealing or others.

The ALNS has a set of destroy and repair neighbourhoods. These define the possible methods of relaxing a solution and of re-optimizing the relaxed solutions. In each iteration, a destroy neighbourhood and a repair neighbourhood are chosen by a roulette wheel selection with probabilities linear to the earlier obtained scores per neighbourhood. With these neighbourhoods, a new solution will be created. The neighbourhood scores are then updated, based on the quality of this new solution. Multiple criteria can be used for the scoring process. Naturally, when a new best solution is found, a high score will be given, but it can

Algorithm 3 Large Neighbourhood Search

$s_{best} = s$ ▷*Initial solution*
repeat
  s' = s
  s' = remove_part_of_solution(s')
  s' = repair_solution(s')
  if $f(s') < f(s_{best})$ **then**
    $s_{best} = s'$
  end if
  if accept(s', s) **then**
    $s = s'$
  end if
until stopping criterion met

Algorithm 4 Adaptive Large Neighbourhood Search

$s_{best} = s$ ▷*Initial solution*
repeat
  $\mathcal{N}$ = roulette_select(π) ▷*select Neighbourhood*
  s' = remove_part_of_solution($s', \mathcal{N}$)
  s' = repair_solution($s', \mathcal{N}$)
  π = update_scores($\pi, s, s', \mathcal{N}, s_{best}$)
  if $f(s') < f(s_{best})$ **then**
    $s_{best} = s'$
  end if
  if accept($s, s', \mathcal{N}$) **then**
    $s = s'$
  end if
until stopping criterion met

also reward not previously visited solutions. This iteration is repeated until some stopping criteria is met. This is shown in algorithm 4.

Iterated Local Search

The ILS was introduced by Lourenço et al. [2010]. The procedure starts with a initial solution s_0 which will be locally optimised to s^*. For every iteration step, it makes a small change or perturbation to the solution s^* to receive solution s'. This solution is then improved by a local search to generate the locally optimal solution $s'*$. An acceptance criterion then defines if s'^* should replace s^*. This process is repeated until a certain termination condition is met. The authors define the basin of attraction as the set of solutions with the same local optimum. The goal of the perturbation step is to jump between these basins of attractions, in order to escape local optima.

Algorithm 5 Iterated Local Search

```
s_0 = GenerateInitialSolution
s* = LocalSearch(s_0)
repeat
    s' = Pertubation(s*,history)
    s*' = LocalSearch(s')
    s* = AcceptanceCriterion(s*,s*',history)
until termination criterion met
```

General Variable Neighbourhood Search

The Variable Neighbourhood Search (VNS) is a local search method which uses multiple neighbourhoods N_k for k = 1,..,k_{max} [Mladenović and Hansen, 1997]. The algorithm works as follows: For each iteration step, set $k = 1$ and create a solution from the k^{th} neighbourhood. Then create x'' by applying some local search method to x'. If x'' is better than x', continue the search with N_k. If not, increment k by one. Do this until $k = k_{max}$, which means that all neighbourhoods have been considered. This process is then repeated until some termination criterion is met as shown in Algorithm 6.

Algorithm 6 General Variable Neighbourhood Search

```
repeat
    k = 1
    repeat
        x' = Shake(x,k)
        x'' = VND(x',k'_max)
        x, k = neighbourhood_change(x,x'',k)
    until k = k_max
until termination criterion met
```

A general implementation of the VNS is given in Hansen et al. [2008]. This uses a variable neighbourhood descent. Variable neighbourhood descent is a fully deterministic VNS as shown in algorithm 7. In the General Variable Neighbourhood Search (GVNS), the shake function selects randomly a solution from the neighbourhood of x and then performs a Variable neighbourhood descent. The GVNS can therefore be seen as an ILS with multiple neighbourhoods in each phase.

4.6. Conclusion

As stated before, combinatorial optimisation consists of finding an optimal object in a finite set. Often, this finite set is the result of discreteness in problems. Routing, scheduling and relocation problems usually have discrete elements and therefore fall under the category of combinatoric optimisation problems. Although for most of these problems, exact solution methods are available, these methods often scale much worse than polynomial.

Algorithm 7 Variable Neighbourhood Descent

```
repeat
    k = 1
    repeat
        x' = select_best_neighbour(x)
        x, k = neighbourhood_change(x,x',k)
    until k = k'_max
until No improvement obtained
```

This non-polynomial scaling causes computational times to grow extremely large with increasing problem size. Finding the global optimum with optimality guarantee is therefore often not possible for real-world problems, and even with increasing computational power this will likely not be possible in the near future. Luckily, real-world problems don't necessarily need the exact optimum. Often a good-enough solution will do as well. Heuristic methods are used to generate these good-enough solutions within a fraction of the time needed to compute the exact optimum.

For both the exact and heuristic solution, a solution method often can be found by analysing similar problems. For this reason, a selection of relevant problems and solution methods were presented in this chapter. These however only give a global view of the techniques used for different problems. In the next chapter, literature research will be done to analyse more specific approaches for problems similar to both the SIRP and the CRP.

5

Literature review

5.1. Introduction

In the previous chapter, a general introduction to optimisation theory was given, with a more detailed view in certain areas. The literature review in this chapter uses that theory. The goal is to give an overview of the literature relevant to the SIRP. Initially, literature in the context of offshore wind farm installation is explored. It turns out that this is not sufficient, and therefore the scope is expanded to vehicle routing research in general.

The CRP will be reviewed after this. It turns out that this is a very unique problem and no direct approaches have been found in literature, although inspiration can be found from areas like container reshuffling or vehicle shunting.

5.2. Offshore scheduling

The literature review starts with looking into optimisation of offshore wind farm installation scheduling. Using a mathematical approach to optimise offshore wind farm installation scheduling is a relatively new field. The first MILP found on this topic was given by Scholz-Reiter et al. [2011].

Scholz-Reiter et al. [2011] considered scheduling in weather conditions with a single vessel, which can install and transport the complete turbine. It solves a scheduling optimisation problem to define the loading and installation choices, with regards to limited ship loading capacity and restricted actions by weather conditions. The problem is formulated as a job-shop problem and solved to minimise the time needed to build the wind farm. Weather conditions are discretised into good, medium and bad conditions and installation and loading actions are restricted on these conditions. The MILP is solved directly by an LP solver for a short time horizon between 1-3 days each time the vessel arrives at the harbour. The reason for this is that weather forecasts are only accurate on short timespans.

A different study by Lütjen and Karimi [2012] includes port replenishment during the installation of an offshore wind farm. A dynamic reactive schedule, as introduced by Mehta [1999], is created. In dynamic reactive scheduling, no initial scheduling takes place, but decisions are made at the moment of execution. At each loading moment, a simple heuristic consisting out of two weighted cost functions is ran to determine the loading set and the order of operations. This cost function takes into account the site status and the weather forecast. A simulation is created to evaluate different rescheduling strategies.

Ait-Alla et al. [2013] presented an MILP model for a medium planning horizon of 2-18 months. The model incorporates chartering costs and weather operation constraints for different vehicles. The planning horizon was split into time periods of one month and for each month, the occurrence of different weather conditions was given. The model then determined which operations should be planned in which month, while minimizing the total costs. The model was solved by an MILP solver.

Irawan et al. [2017] used the same weather conditions as Scholz-Reiter et al. [2011]. Based on these conditions, feasible timeslots are generated for each vessel. Each timeslot defines a set of consecutive days in which a certain action can be executed and is determined by the weather and the weather installation criteria. The optimisation problem then is to assign tasks to these timeslots. This is minimised for total installation costs with an MILP model.

A heuristic algorithm is used to optimise for both total cost and completion period. This bi-objective optimisation calculates solutions for both objectives. A solution is then searched where the sum of the distance to both objective-solutions is minimised. This is done both for a VNS and SA algorithm where the VNS is shown to outperform the SA algorithm.

In the field of offshore wind farm installations, no literature was found to optimise the routing process, nor anything that takes into account synchronization requirements while using multiple ships for installation. For this reason, the literature research is expanded to optimisation in offshore wind farm maintenance scheduling.

In this topic, a routing approach is used by Zhang [2014] who gives an MILP model which minimises the travelling costs with a penalty for delayed maintenance. They use a modified ACO algorithm where they use a group of ants per vessel and remark that it is a time consuming technique, but that it has the benefit of allowing non-linear components. Furthermore, they model the weather by setting a maximum amount of working hours per day for each vessel.

Stålhane et al. [2015] presented research on optimal routing of maintenance while minimizing transportation, downtime and delay-penalty costs. They present two MILP models, an arc-flow and a path flow model. In an arc-flow model for routing, each arc in the network is a variable. In the path flow model, each vessel route is a variable. This reduces the search space, since every solution in the path-flow model is a solution in the arc-flow model, but the opposite is not true. Since the numbers of paths becomes very large (factorial with amount of nodes) the paths have to be generated while doing the optimisation. This is done by using a heuristic column generation approach in a BC algorithm. The main principle of column generation was introduced in Section 4.5.1. A heuristic column generation uses a heuristic algorithm to define which columns will be generated. The path-flow model gives near optimal solutions at a significantly smaller computational time than the arc-flow model.

5.3. Vehicle routing Problem

Maintenance scheduling points us in the direction of the vehicle routing problem. This is a classic optimisation problem and was introduced in Chapter 4. It was stated here that the VRP shares the same difficulty of preventing subtours as the TSP. In the remainder of this section, first the main solution methods for solving the VRP will be presented. These main solution methods are aimed at the general VRP, while the SIRP has different characteristics. Therefore, generalizations are studied after this. Generalizations modify the VRP problem to also account for other aspects.

5.3.1. Solution Methods

The large variety in VRP extensions results in a lot of different solution methods. A classification of this is given in Ismail et al. [2017] and is used as a basis for figure 5.1. As in every optimisation algorithm, there is a distinction between exact algorithms and heuristic algorithms. Since the scope at this moment is the general VRP, we look at the most studied extensions of the VRP: the Capacitated Vehicle Routing Problem (CVRP) and the Vehicle Routing with Time Windows VRPTW.

It can be seen in Figure 5.1 that there are three methods to find an exact solution. These different approaches specify how to handle the prevention of subtours. These are: Branch and But, Branch and Price and introducing extra variables. For the heuristic approach, a distinction is made between classical heuristics and meta-heuristics, although in literature this line might not always be very clear. As defined earlier, we see classical heuristics as problem-specific. An example of this is the nearest neighbour algorithm, where a vehicle always goes to its nearest unvisited neighbour. Metaheuristics are higher level heuristics

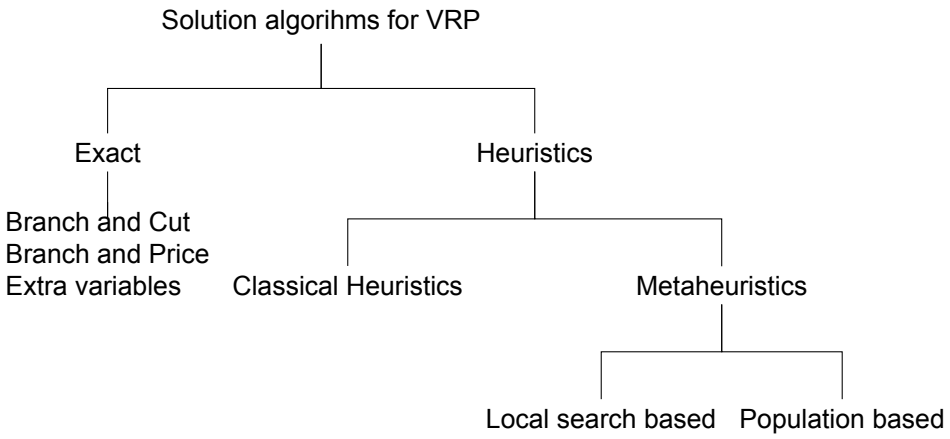


Figure 5.1: Classification of solution procedures for VRP

which can be applied to different problems. An example is tabu-search or simulated annealing, both explained in Chapter 4.

Exact solution methods

A good overview of recent exact solution methods is given by Baldacci et al. [2012]. The main solution method used in nearly all exact approaches for the VRP is solving an MILP model. As in the TSP, the main problem for the VRP is the elimination of subtours. In the VRP, the amount of subtour elimination constraints increases exponentially, and therefore defining all constraints in the original MILP is not possible. Therefore, 3 different methods were developed as shown in Figure 5.1.

Branch cut

As presented in Section 4.5.1, a BC algorithm adds constraints during the branching process. It consists of a formulation and a solution to the separation problem. The separation problem was described earlier in Section 4.5.1 and consists of finding an inequality which separates the current infeasible solution from the solution space. This is repeated until the solution lies within the feasible polytope. If this solution is not integer, branching occurs and the process is repeated until an integer optimal solution is found.

The first BC-algorithm for adding subtour elimination constraints was not given for the VRP, but instead for the TPS. Padberg and Rinaldi [1991] proposed a BC algorithm to solve this. At the beginning of this paper, the 'classical' approach for solving the TSP is given. This basically consists of solving the MILP for a subset of all inequalities, solving the separation problem, adding these inequalities and restarting the algorithm. They note that this method wastes resources by restarting the MILP solver multiple times and therefore apply a Branch and Cut algorithm to the TSP.

Subsequently, Augerat et al. [1998] implemented a BC-algorithm for the symmetric VRP. With this approach, a 135-customer instance was solved to optimality, at that time the largest in literature.

The formulation of Augerat et al. [1998] consists of multiple types of subtour elimination constraints. The first are the capacity constraints Equation (5.1), where $D(S)$ is the vehicle-demand of S. This equation then says that for each subset S of demand nodes (excluding depot), the amount of arcs entering and leaving S is at least the amount of vehicles needed to service S times two. When there is a subtour, there is at least one S where this constraint is violated. A lower bound for the vehicles to visit S, can be set as $\lceil d(S)/C \rceil$ where d is the demand for S and C the capacity per ship.

$$\sum_{i \in S, j \notin S} x_{ij} + x_{ji} \geq 2D(S) \forall S \subseteq V_0 \tag{5.1}$$

Furthermore, Augerat et al. [1998] gives another heuristic for the capacity constraints separation called the greedy shrinking algorithm, which expands an initial set S by iteratively adding the node v which maximises $x(S : v)$

Branch and Price

One method to overcome the exponential amount of sub-constraints is using a Branch and Price algorithm. For the VRP, a Branch and Price algorithm uses set partitioning. Set partitioning is the grouping of elements into subsets, such that each element appears in exactly one of the subsets. A formulation of SP for the VRP was originally proposed by Balinski and Quandt [1964]. The idea of this was the following: Let each possible route be a variable with a corresponding cost. Then, the optimal solution can be found by selecting the minimum costs subset-combination in which each node is included exactly once. This is shown in Equation (5.2). Here, R is the set of all routes and N the set of all nodes, with 0 being the depot node. θ_{ir} is a binary parameter which is 1 if node i is in trip r and zero otherwise. c_r is the cost of trip r and the variable x_r defines whether trip r is included in the solution.

$$\min \sum_{r \in R} c_r x_r \tag{5.2a}$$

subject to

$$\sum_{r \in R} \theta_{ir} x_r = 1 \forall i \in N \setminus \{0\} \tag{5.2b}$$

$$x_r \in \{0,1\} \forall r \in R \tag{5.2c}$$

By taking only valid trips in R without subtours, subtour elimination constraints are not needed and the model will only consist of $|N \setminus \{0\}|$ constraints. The trade-off is an exponential number of variables. This means that the formulation in Equation (5.2) still cannot be solved by a regular BB algorithm. To overcome this, a subproblem will be defined. This subproblem describes the trips in R which have the possibility of improving the solution.

Desrochers et al. [1992] proposed a BP algorithm to solve a set partitioning formulation of the VRPTW. As show in chapter 4, the Branch and Price algorithm is a variation of a Branch and Bound algorithm which creates variables during the branching process. The BP algorithm has the best performance for VRPTW with tight time windows. For less constrained problems, or even without time windows, the BP algorithm will perform much worse and will quite often fail to find a solution for problems which would be solvable with smaller time windows [Fukasawa et al., 2006].

This improvement in efficiency can be explained by the amount of candidate trips. A trip is a candidate if it has reduced cost. Reduced cost means that adding a certain route might decrease the total cost. Consider a vehicle at a certain node which has to select a node to travel to. If the new node has a time window opening far in the future, it means that if the vehicle travels there, it has to wait at that node. This increases the costs, and makes it less likely that a route with this arc has reduced cost. If the time window is in the past, it is not even possible to visit that node. This limits the amount of candidate routes for the BP algorithm, and therefore increases the performance.

Without this improved performance, generating columns was not a very efficient approach for the VRP without time windows. Fukasawa et al. [2006] overcame this problem by combining BP and BC. This is then called Branch and Cut and Price (BCP). In this approach, an arc-flow formulation P_1 is created and a route-flow P_2 which is linked to the arc flow by defining the routes as sets of arcs. An algorithm then uses both column as row generation to optimise the intersections of both problems.

Heuristics solution methods

It was shown earlier in Figure 5.1 that for heuristic methods a division was made between classical heuristics and meta heuristics. Classical heuristics are problem specific and often

use real-world characteristics. An example of these classical heuristic methods is the sweep method, introduced by Gillett and Miller [1974]. This algorithm uses the polar coordinates of the nodes in a plane with the depot at the origin, to divide the nodes in clusters. This clustering is done by ordering the nodes by increasing angle, creating clusters as shown in Figure 5.2. This algorithm thus uses the physical locations of nodes. This requires the assumption that these physical locations are known and that the distance between two nodes is representative for the cost of the arc between them. In more abstract networks, this might not always be the case.

There are multiple other classic heuristic solution methods. However, since the SIRP includes an abstract network where arc costs include things like installation time or preparation times, the physical distance between two nodes might not always be representative for the arc cost between two nodes. For this reason, further research is aimed at metaheuristics instead of classical heuristics

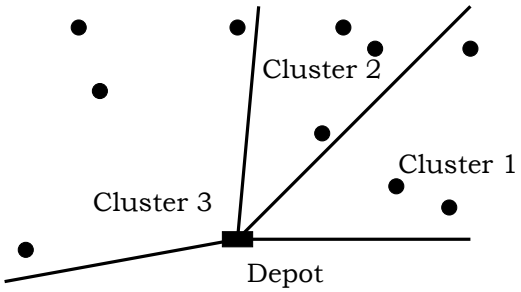


Figure 5.2: Sweep algorithm clustering

Metaheuristics can be classified into two groups: Single agent heuristics and population-based heuristics. Single agent heuristics use a single agent which explores the solution space, like for example tabu-search or simulated annealing. The single agent in these algorithms move between feasible solutions. In a classical local search, the agent only moves to a new solution when that solution is better than the current one. This means that there is no option in escaping local optima. To overcome this limitation, simulated annealing and tabu search algorithms were developed. These were presented in Section 4.5.2.

Both simulated annealing and tabu search are relatively old techniques. Tabu search was introduced for the VRP by Glover [1977], and the earliest found simulated annealing implementation for the VRP was that of Alfa et al. [1991]. Population based algorithms, often inspired by nature, are more divers, but in general consist of multiple agents exploring the search space simultaneously. The difference in population based algorithms usually lies in how the different agents explore the search space and how they influence each other.

Although population based algorithms are also relatively old, the first genetic algorithm for the VRP which could compete with the tabu-search was presented by Prins [2004]. A genetic algorithm, as presented in Section 4.5.2, includes a population of feasible solutions. As explained in Chapter 4, a GA uses the principles of selection, reproduction and mutation. The novelty of Prins [2004] was found in the mutation step. Instead of just performing a single mutation, a local search algorithm was used to optimise each child.

Multiple other population search methods have been explored. Bell and McMullen [2004] implemented one of the first well-performing ant colony optimisation (ACO) algorithms for the VRP. Here multiple ants are modelled. Each ant starts at the depot and selects a node to travel to. The choice of which node to select is stochastically based on the amount of pheromone on each arc from the current node. This process is repeat until all nodes are visited. If the capacity of a tour is exceeded, a return trip to the depot is added before visiting additional nodes. When all nodes are visited, pheromone is deposited on the selected arcs, with more pheromone per arc on shorter arcs. This motivates other ants to also take these arcs.

Many different solution techniques have been implemented for the vehicle routing problem. However, most modern literature is not aimed at finding the best algorithm for the classical VRP. Instead, most authors study different generalizations. 327 Articles regarding the VRP were studied by Braekers et al. [2016]. From these, only 16 articles considered the VRP in its classical form. This variety of problems has resulted in a variety of solution approaches. Therefore, different aspects of the SIRP will be evaluated in the next section, along with relevant literature research. The question is not: what is the best algorithm for

solving the VRP, but what are the best algorithms for VRPs with similar aspects to the ship installation routing problem?

5.3.2. Generalizations

Although already quite useful in its original formulation, real life problems often require generalizations. A generalization is an extension to include other characteristics of the problem.

Many generalizations have been introduced. Braekers et al. [2016] give an excellent survey on different extensions. Although already included in the original formulation, the Capacitated Vehicle Routing Problem specifically states the version of the VRP with capacitated vehicles. This distinction is made, since other versions without capacitated vehicles exist. However, the CVRP is by far the most studied version. It has been included in over 90% of all articles considered in the survey. It is quite clear that this extension has great applicability in real life routing problems, as it is quite common for vehicles to have a limited capacity.

The other main generalization is the introduction of time windows to create the VRPTW. Next to these, there are several other characterizations which have received moderate attention. These are backhauls or pickups, multiple depots and heterogeneous vehicles. An extension interesting for our research is the introduction of precedence and coupling constraints. This means that certain vehicles have to visit a node before another vehicle, or that they have to perform certain acts simultaneously.

5.3.3. Heterogeneous Vehicles

The classical VRP uses a set of homogeneous vehicles. However, in the SIRP, multiple ships are used with different characteristics. The use of vehicles which differ in load capacity and compatibility is referred to as the use of heterogeneous vehicles. This was first implemented by Golden et al. [1984], who created an algorithm which first created a grand tour which visited all customers before partitioning this into segments.

For exact formulations, an elaborate paper was published by Yaman [2006]. This paper introduced 6 different formulations and various cutting planes. The first four formulations where based on subtour constraints from Miller et al. [1960]. The first formulation uses a single variable x_{ij} if there is any vehicles traversing the arc (i,j) and another variable a_{ik} which links the ending nodes of a route to a specific vehicle.

The second formulation introduces a variable which links the other nodes in the route to a specific vehicle. In the third formulation, y_{ijk} replaces x_{ij} in the first formulation, being equal to one if vehicle k traverses arc (i,j). This allows for different costs per vehicle. The same substitution from the second formulation two results in the fourth formulation.

Furthermore, two flow formulations are given, where the capacity and subtour constraints are expressed according to the model of Gouveia and Pires [1999]. This uses a variable f_{ij} to specify the flow on arc. The flow on an arc can be seen as the amount of supplies transported along that arc (i,j). Similar to the transition of x_{ij} to y_{ijk}, the last formulation is created by transforming f_{ij} to g_{ijk}. In addition to these formulations, for each formulation multiple cuts from the homogeneous Vehicle Routing Problem were adapted to the heterogeneous vehicle routing problem.

According to Koç et al. [2016], most heuristic approaches in the heterogeneous vehicle routing problem are tabu search algorithms. Euchi and Chabchoub [2010] use a tabu search with two moves. An exchange move where two nodes are exchanged between routes and an insert move where a single node is inserted into another route.

A similar approach is taken by Brandão [2011], who also uses a tabu search. Besides the moves used by Euchi and Chabchoub [2010], it also uses a 2- and a 3-node insertion move. A cycle based move selection mechanism is used to consider a move i only at every n_i iterations, therefore reducing the computational needs.

Tarantilis et al. [2004] use a list based local search method. Here multiple solutions are stored in a list and local search is implemented to improve these while gradually tightening the acceptance criterion like in a simulated annealing algorithm. A quite different approach,

although still local search based, is given by Naji-Azimi and Salari [2013]. Instead of the traditional vehicle routing moves, they define the neighbourhood based on an MILP based improvement procedure. Each iteration, a set of routes is removed, and replaced based on the result of an MILP.

5.3.4. Synchronization

Although the CVRP and the VRPTW turn out to be good starting points, some modelling parts in our problem are still missing in this. One of these parts is synchronization. This means that certain visits are synchronised in space and time. This is necessary, because sometimes ships need to carry out an action simultaneously, or with a precedence relationship. This happens for example during transportation of monopiles, or when making sure that the transition piece is only installed after the monopile has been installed.

Drexl [2007] introduced the Vehicle Routing Problem with Trailers and Transshipments (VRPTT). This is a generalization of the VRP which include autonomous (lorries) and non-autonomous (trailers) vehicles. The non-autonomous vehicles can only travel on arcs if attached to autonomous ones. There is no compatibility constraint between vehicles, i.e., any trailer can be transported to any lorry. Next to the standard customer covering constraints (all customers must be visited), three more synchronization constraints between vehicles are implemented in this problem: Spatial, temporal and load synchronization. Spatial and temporal synchronization mean that vehicles have to be at the same time or location, respectively. Load synchronization is the transfer of load between vehicles.

Complex networks are used for this problem. Where a classical VRP has nodes representing physical location and arcs representing travelling between these, the VRPTT and other rich VRP might need networks where nodes represent combinations of space, time, vehicles, operations etc. An example is the transshipment of load between two trailers. There will be a vertex for the transfer-location at the beginning and one at the end of the load-transfer. The arc between these two vertices corresponds to the loading time.

Drexl [2007] gives three MILP formulations for the VRPTT. The first formulation uses a turn based graph. This is a graph which has, next to a set of vertices and arcs, also a set T with turns. A turn is a set of two arcs which can be traversed directly after each other. Drexl [2007] introduces turn penalties: costs of taking a certain turn. The concept of turn penalties make it possible to put costs on traversing an arc based on the previous arc and make certain sequences impossible by setting the turn penalty to infinity. The turn variables x_{khij} are introduced, which are equal to one if vehicle k traverses arc (i,j) directly after (h,i). For edge synchronization, lorry-trailer variables $x_{kk'ij}$ are used which are equal to one if lorry k pulls trailer k' over edge (i,j).

The second formulation is an arc based one. It uses a time-space-operation-vehicle network. To handle multiple time windows, a vertex is added per customer per time window. Tasks and operations are bound to vertices, with some split into two vertices connected by an arc. The routing part is then simply done by the variables x_{kij} which are equal to one if vehicle k traverses arc (i,j).

The same graph is used with the path-based formulation. A Branch and Price formulation is suggested with a master problem and a subproblem. Each variable defines a possible vehicle-route. These routes are generated in the subproblem. The master problem consists of selecting a subset of these variables to include in the optimal solution. Drexl [2007] calls a constraint coupling if it combines variables from multiple paths, and puts all coupling constraints in the master problem and all non-coupling constraints in the subproblem. These coupling constraints are, besides node-covering constraints, timing and synchronization constraints.

The columns in a BP for the VRP are generated by a path-search algorithm [Feillet et al., 2004]. This generates an elementary shortest path with resource constraints (ESPRC). Elementary means that each node is visited only once, and resource constrained means each arc requires a set of resources, and a path is only feasible if the amount of resources required is below a certain value. The resources used while traversing an arc are defined by the resource extension function (REF). In Desrosiers [1998], two highly desired properties are listed for the REF. First of all, every REF for an arc (i,j) should only depend on the resource vector at i,

secondly every REF should be non-decreasing. Drexl [2007] states that the required REFs for the VRPTT problem do not have these properties. It is concluded that at the time of writing, there was no valid Branch and Price approach available for the VRPTT. This statement is repeated in Drexl [2012].

Bredström and Rönnqvist [2008] studied the combined vehicle routing and scheduling problem with temporal precedence and synchronization constraints for home healthcare staff scheduling and forest operations. While both applications have the same characteristics, it is especially easy to see how forest operations resembles the offshore installation problem. Trees are cut and processed by a harvester, after which they are picked up and transported by a forwarder. Some transporters can load independently, some need to be accompanied by a truck with a crane.

A network representation is given. Nodes represent physical locations and service times are specified for nodes. An edge only represents traveling between locations. It is also stated that when there are locations which can be served by specific vehicles, there will be vehicle dependent networks. They formulate an MILP model with routing variables and scheduling variables. The routing variables are based on a VRP problem with an arc-flow formulation. They refer to the constraints for visits per node, synchronization and precedence relationships as complicating constraints because they couple the otherwise independent variables. Most vehicle routing only have the visits per node as complicating constraints and this shows the increased difficulty for the SIRP.

The MILP model was solved to optimality in CPLEX for instances up to 20 nodes. For larger test instances, a heuristic was developed. This heuristic includes a penalised dummy variable for nodes not visited by any vehicle by adjusting certain constraints. The algorithm then creates a restricted model by associating each node with a set of vehicles which are allowed to visit it. This restricted problem is then solved by first solving the LP-relaxation, removing all arcs not utilised in this solution, and then finding a feasible solution to the restricted MILP problem. Subsequently, an iterated process is started. For each iteration, the set of included arcs is slightly modified and the relaxation and feasible solution steps are repeated. This is done until a certain time limit is exceeded.

Drexl publishes a survey on synchronised VRPs in Drexl [2012]. In this, among other things, the algorithmic issues for heuristics are discussed while dealing with synchronization. He states that LS based methods is the main heuristics for large-scale VRP problems. This procedure exploit the fact that it is possible to independently change a very small set of routes. This is true for the standard vehicle routing problem. When a route is considered, making a small change (like swapping two nodes) results in a new feasible solution. However, Drexl notes, this is not true for synchronised VRP. After a small modification it might be possible that the synchronization constraints cannot be fulfilled anymore. This can be showed with an example: Consider vehicles A and B. If A and B have to visit each node simultaneously and A has route $(0, 1, 2)$ and B has route $(0, 2, 1)$, A will be waiting at node 1 until B arrives there, but B will be waiting at node 2 until A arrives there. Therefore, this route is infeasible.

Furthermore, even if the resulting route after a move is feasible, the unchanged routes were optimised for the original route. Since this is not the case for the new route, there is a large chance that the new solution has very long waiting times. This results in a large probability for the local search algorithm to label the new solution and thus the move as bad quality. However, this is largely influenced because the other routes were synchronised with the original route, and not by the move itself. Therefore, due to the synchronization constraints, a good move might be labelled as bad by the local search algorithm. Drexl [2012] therefore concludes that most VRP algorithms are not suitable for the synchronised VRP.

Drexl [2012] also provides a classification of synchronization constraints. The constraints are classified in the categories task, operation, movement, load and resource synchronization. Tasks are mandatory duties which have to be fulfilled by zero or more vehicles. In the classical VRP, the task can be stated as all customers having to be visited by at least one vehicle. An operation is something that must be performed by a vehicle at a vertex to allow the completion of one or more tasks. This may induce dynamic time windows. Operation

synchronization puts constraints on allowed time between operations. This means that it might imply that certain operations have to be done simultaneously, or one after the other, etc. The third synchronization mentioned was movement synchronization, which refers to non-autonomous vehicles not being able to travel without assistance of autonomous vehicles. Load synchronization includes the transfer of load between multiple vehicles and resource synchronization models vehicles competing over a common scarce resource.

Operation synchronizations relevant to the SIRP are direct synchronization and synchronization with precedence. For exact methods, multiple column generation approaches were suggested by Grünert and Sebastian [2000] and Crainic et al. [2009], although these include only a suggested direction and no implementation. The main heuristic method for precedence synchronization, noted in Drexl [2012], was to explicitly determine leg sequences. A leg can be seen as a part of the route without synchronization.

Lu and Dessouky [2006] uses an insert-based heuristic for a pickup and delivery problem with time windows. An insert-based heuristic starts with an empty solution and inserts nodes. This normally picks the best node for the best location based on smallest increased costs. Lu also considered the change in slack time. This is the time between visiting a node, and the nearest closing of the time window for that node. By using this as a criterion for inserting it leaves more space in the time window for other insertions.

Salazar-Aguilar et al. [2013] introduce the synchronised arc and node routing problem. The difference with previously given synchronised problems is that the requirement is now that all arcs have to be visited, instead of all nodes. They implemented an application of this problem to a road marking network. Here, they consider painting vehicles and replenishment vehicles. Painting vehicles have to traverse all arcs in the network while replenishment vehicles have to synchronise with the painting vehicles to make sure they do not run out of paint. This problem was solved with an ALNS.

More research on the home-healthcare routing problem was done by Ait Haddadene et al. [2016]. Nursing staff has to visit patients at their homes. Since there might be actions which have to happen simultaneously, or actions which have to occur after others, synchronization is important. The problem is formulated as a VRP with synchronization constraints and solved by a two phase heuristic. The construction phase consists of creating an initial solution with GRASP, and the second phase consists of optimizing this solution with an ILS method. These methods were explained in Section 4.5.2. The basic idea of ILS is that after each move in the local search method, a new optimisation algorithm starts to repair the synchronization constraints.

5.3.5. Multitrip

Another aspect on which the SIRP differs from the general VRP is that vehicles are allowed to do multiple trips. We define a trip as the path of a vehicle between leaving from and returning to the depot without any intermediate stop at a depot. A route is the complete path of a vehicle, consisting of multiple trips.

An overview of mathematical notations for the multitrip vehicle routing problem can be found in the survey paper Cattaruzza et al. [2016]. The most common formulation is a 4 index formulation $x_{krij} \in \{0,1\}$ which is equal to 1 if and only if arc (i,j) is in trip r of vehicle k. A limitation of this notation is that the maximum number of trips has to be set and for each possible trip per vehicle, $i * j$ variables are added. This will lead to a weak formulation if there is no tight upper-bound on the number of trips.

A three-indices formulation was introduced by Aghezzaf et al. [2006]. The binary path variable x_{kij} is used to define if arc (i,j) was used by vehicle k. The variable f_{kij} was added to include the capacity constraint. This variable is equal to the load of vehicle v after point i when moving to point j, or 0 when arc (i,j) is not used by vehicle v. Thus, the load of a vehicle decreases along the trip, and by constraining the load to be larger than zero for every index-combination, the capacity requirements are enforced.

The same x variable was used by Buhrkal et al. [2012], with a variable for cumulative demand instead of load. This is the inverse of a load variable. The load variable starts with the total capacity at the depot, and is reduced by the node-demand for every node in the trip. The cumulative demand variables starts at 0 at the depot and is increased with the node demand at every node it visits. Buhrkal et al. [2012] study an optimisation problem to collect waste and also include time windows in this. The problem is defined as semi-multitrip, because each vehicle visits multiple waste-disposal sites. Like multiple other authors, they include the timing variable t_{ik} to define the time at which vehicle k visits node i. This notation requires the constraint that every node is visited only once per vehicle, since otherwise it is not possible to define a single time for that node-vehicle combination. Therefore, the notation used is only applicable to a semi-multitrip problem with multiple depots, but where each depot is visited only once.

Two authors were found using a 3-index formulation without the vehicle index. But although, since it is quite clear that this formulation will not be compatible in problems with heterogeneous vehicles and vehicle synchronization constraints, this notation will not be discussed further. The articles in which this notation was found [Hernandez et al., 2014; Azi et al., 2007] did not include synchronization constraints nor heterogeneous vehicles.

Two 2-index formulations have been found for the multitrip VRP. Although previously stated that a vehicle index is desired, looking into these can provide insight in how to handle other problems, specifically how to include trip times. The problem with trip times is that most time formulations are t_{ik} formulations, similar to Buhrkal et al. [2012]. This only allows for a single visiting time per vehicle-node combination. This does not allow for multiple visits to the depot, and thus not for multiple trips with a single depot.

A solution for the timing problem can be found in Koc and Karaoglan [2011] who use a 2-index formulation with only $x_{ij} \in \{0,1\}$ equal to 1 if and only if arc (i,j) is used by any vehicle. They use a t_{ik} variable for the visits time for all nodes except for the depot. The variable $w_{ij} \in \{0,1\}$ is introduced, which is equal to 1 if and only if the trip ending in node j is followed by a trip starting at node i. The standard timing constraints based on x are then set for all nodes except the depot nodes. Furthermore, timing constraints are added based on w_{ij}, setting the visiting time of j later than the visiting time of i, plus the cost of travelling from i to the depot, and the cost of travelling from the depot to j. Thus, all times of the first nodes in a trip are not set based on x, but based on w. This approach can be seen as not specifying the depot times, but instead linking directly from the last visit in a trip to the first visit in the subsequent trip.

Rivera et al. [2015] uses a formulation with two types of arcs: Standard arcs and replenishment arcs. When a vehicle has no capacity left, a replenishment arc must be used after which the capacity is reset again. These replenishment arcs represent returning to the depot. This method is mathematically equivalent to Koc and Karaoglan [2011].

The earliest found heuristic approach for the multitrip VRP is given by Taillard et al. [1996]. They use a two-phase algorithm which consists of firstly generating a list of trips and secondly selecting trips to create a solution. The creation of the list of trips was done with a tabu search algorithm presented in Taillard [1993]. The neighbourhood was defined as moves of selecting two routes and swapping or inserting up to 2 nodes. The second phases then constructs a solution by selecting several trips as a bin packing problem. In bin packing problem, multiple objects have to be packed into a number of bins with a finite capacity. This approach was expanded by Petch and Salhi [2003] by adding a third step. This step contains an LS with a move that splits a trip in two and then move one of the two resulting partial trips to another vehicle. Subsequently, another local search improves the solution by swapping and inserting customers.

A Tabu Search algorithm for the multitrip vehicle routing problem was implemented by Alonso et al. [2008]. The neighbourhood consists out of two moves. A traditional insertion move where one customer is removed from a route and inserted at another, and additionally a problem specific move based on accessibility restrictions per node.

Besides these single-agent local search methods, multiple population based heuristics

were found. Salhi and Petch [2007] uses a genetic algorithm to divide customers into clusters. A solution is then created by generating a single route per cluster, and this solution is further optimised by a local search heuristic. Cattaruzza et al. [2014] also use a genetic algorithm and introduce the term of combined local search. This is a repair procedure for solutions with an increased cost based on the previous solution. By swapping routes, it can turn the solution with an increased cost to a solution with a decreased cost. This is possible due to the time restriction characteristics of their specific problem. .

5.3.6. Multiple Installation options

Multiple installation options add multiple extensions to the VRP. Ignoring the time constraints, an installation method can require multiple vehicles to visit the same node. Per installation method, multiple sets of vehicles are defined, and one vehicle from each set has to visit. With multiple installation method, each installation method is a set of set of ships and it must hold that for one set of sets a single vehicle from each set must visit a node. Unfortunately this has not been found in literature.

5.4. Component Relocation

The relevant literature for the crane optimisation is less abundant than for the ship routing problem. To the authors best knowledge, no crane optimisation model exists for relocations in a grid with spatial blocking constraints. However, relevant other problems were found. The most studied problem which contains similarities with the CRP is the container relocation problem in harbours. Other relevant research was found in the field of train shunting and factory item relocations.

5.4.1. Container relocation

The container relocation problem consists of minimizing crane movements during the handling of containers. It has quite some similarities with the turbine component relocation problem, although the blocking constraints are simpler. Shipping containers are used for transport on ships, trains and trucks worldwide. In between journeys, they are often stored in container terminals. Cranes are used to relocate them based on arrival and departure requirements, similar to the TP crane optimisation.

Containers are placed in stacks and cranes are able to grab containers from above, as seen in Figure 5.3. This eliminates the spatial constraints based on neighbouring containers, but due to the stacking, it adds the constraint that only containers at the top can be relocated. Lehnfeld and Knust [2014] published a survey identifying the different types of container optimisation problem. They categorise three types of problems. The first problem consists of choosing the storage location during unloading from a ship to the storage location. The second type is the pre-marshalling problem prepare containers for loading at a later time. The third type consists of loading from storage location to the ship.

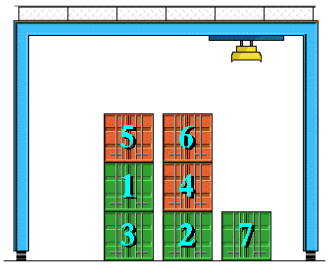


Figure 5.3: Container Relocation

Furthermore, Lehnfeld and Knust [2014] note that most research is done considering just one of these problems. The only author listed who published research on a combined problem is Malucelli et al. [2008]. They present a dynamic programming method, but only for a single stack of items. Also in similar surveys like Carlo et al. [2014] and Vis and De Koster [2003], no optimisation is found which handles both unloading, loading and pre-marshalling.

Wan et al. [2009] give an MILP formulation for emptying a set of container stacks. The main variable used is x_{ticp}, determining the position p of container i at timestep t and column c. It thus defines if a container is at a certain space at a certain time. The other variables used indicate if a container blocks another one, are in the same column, is reshuffled at a

certain time and if it reshuffled at the same time with another container. The formulation depends strongly on the 1-dimensional blocking constraints, caused by container stacking.

Additionally, they introduce multiple heuristic approaches. The best performing one is a rolling horizon method which uses the MILP formulation to retrieve the first k containers. Subsequently, the moves for retrieving the first container are stored and the MILP formulation is used to retrieve the first $k+1$ containers, and so on until all are received. They show that this heuristic outperforms multiple other heuristics different authors.

A more versatile MILP formulation is given by Caserta et al. [2012], who studied the same problem of retrieving a fixed set of stacks in a set order. Their version of the problem holds the assumption that at retrieval of a container, only containers in the same column can be relocated. They note that although this assumption reduces the search space, it might cut away optimal solutions. The formulation given uses a binary variable which indicates if a certain component moves between two positions at a certain time, and variables which indicate if a container is retrieved or available. They also prove $\mathcal{NP}$-hardness of this problem.

A stronger formulation was given by Petering and Hussein [2013], who used a variety of decision variable. They combined moving variables (If a container does a certain move) and location variables (if a container is at a certain location), and split up the moving variables in decision variables for moving between stacks and inside the same stack. The variables for movement in a stack are based on distance from the top, meaning that when a container is placed on top of a stack, all containers below move one position away from the top. Furthermore, they give a heuristic algorithm. This algorithm handles the relocation in certain steps by allowing and disallowing certain moves. They implement a blocking based approach, which determines if a container blocks another one based on looking ahead to when they should be retrieved. If a component i is above a component j and j has to be removed before component i, component i is said to be blocking.

So far these all these consider a fixed retrieval order. Kim and Hong [2006] studied the problem of departure in batches. Ordered groups of departures are given and within these groups the order is free. A problem specific heuristic is given, but no MILP formulation. The same holds for the research of Forster and Bortfeldt [2012].

5.4.2. Train shunting problems

Another relevant problem is found in train or tram shunting. This was studied by Winter and Zimmermann [2000] for trams. Trams have a daily schedule, and arrive at the end of the day to a storage yard of multiple parallel tracks with one dead-end. The trams stay here overnight, and in the morning they leave again in a certain order. Since not all trams are suited for the same route, and because there is limited space in the storage yard, it is important to position trams correctly such that on arrival they are not blocked by other trams. Winter and Zimmermann [2000] introduces four dispatch problems. Firstly the minimum shunting problem is introduced which minimises the amount of shunting done in the yard. Shunting is matching separate tram or train units into complete trains. Secondly, the minimum type mismatch problem was formulated. In this problem, trams are allowed to fulfil routes not originally assigned to them, and the amount of these mismatches is minimised. For both problems, two similar problems are defined by just considering the departure. For all problems MILPs are given and multiple heuristic approaches.

Haahr et al. [2017] studied the problem of parking trains unit on parallel tracks, with a fixed arrival and departure schedule. The tracks have one dead end, thus trains arrive and leave from the same direction. The goal is to assign each arriving train unit to a track such that on departure, no train unit will block it. Reshuffling is not allowed. Haahr and Lusby [2017] gives some feasibility checks to see if solving the problem is possible, and furthermore present three solution methods. The first one is based on constraint programming. In constraint programming, logical relationships are defined between variables. For example, x and y cannot have the same value. A set of possible values is defined for each variable, and the logical relationship prune these possibilities until a solution is found which requires all relationships.

Additionally, a solution method is given which uses column generation. In this approach each variable represents a matching pattern for a single track. This matching pattern holds a feasible schedule for the trains arriving and departing for that track. A matching is feasible if all arriving trains leave the track and the first in first out principle is satisfied. The master problem is then solved to select a single matching pattern per track such that as many trains as possible are included.

The third solution method is a randomised greedy construction heuristic. This heuristic assigns the arriving trains to tracks based on the current occupation of trains. This simple greedy algorithm does not always result in a solution, but since it takes a very small amount of time so it can be run many times until the time limit is exceeded or a feasible solution is reached. This method thus uses the principle of using a simple algorithm with a chance of getting a feasible solution, and trying this many times.

5.4.3. Factory item relocation

In factories where handling of stacked objects occur, it might be possible that the order of delivery and retrieval differ. This results in the need of reshuffling. For most purposes, this amounts small instances which can be solved manually, but in some industries optimisation is proven to be profitable. Mathematically, the problem of handling components in factories is nearly identical with handling containers in a terminal.

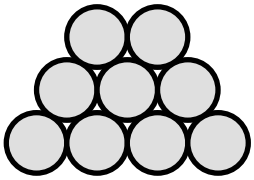


Figure 5.4: Coil Stacking

Kamrad et al. [2012] study two types of item shuffling: plate shuffling and coil shuffling. Plate shuffling consists of shuffling stacks of metal plates which are needed in the steel production industry. The movement of these plates is identical to containers in a terminal. An MILP is given for this problem based on location variables and variables which define if a certain item is shuffled to a certain stack at time t. A similar model is given for the coil stacking problem. This handles coils to be stacked as shown in Figure 5.4.

They produce an algorithm which generates a lower bound by looking at the reshuffles needed per retrieval, but ignoring the effect of these reshuffles during later retrievals. Two types of heuristics are developed, a tabu search and a rolling horizon method. It is noted that the tabu search performs better when crane travelling distance is considered and the rolling horizon performs better when crane travelling is disregarded.

Lu et al. [2016] optimise the plate storage problem. In this problem, plates will arrive in a given sequence and they will have to be stored in stacks. A retrieval period is known, and they have to be places as optimal as possible to minimise reshuffling moves during retrieval, although this phase itself is not modelled. They define plates as blocking if there is at least one plate below it which has to be retrieved earlier.

An MILP formulation is given which has the following binary decision variables which are equal to one when: x_{is} if plate i is piled on stack s, y_{ij} if plate i and j are piled on the same stacks and b_i if plate i is blocking. This is solved with an MILP solver. Multiple strategies are discussed to speed up this process.

5.5. Conclusion

For the SIRP it has been showed that the literature directly from the area of offshore installation was limited in quantity and not applicable to the SIRP. Slightly more relevant research was found in maintenance scheduling, although this still proved to be insufficient. For this reason, the literature research was expanded to research about VRPs with common characteristics as the SIRP.

In the VRP capacity constraints on the vehicles are a very common requirement, available in most VRP formulations. The most used exact solution approach was BP or BCP, where each variable represents a route. For heuristics, nearly all algorithms used were variants of local search, with a few exceptions like ant-colony algorithms. For single agent local search based methods, tabu-search was used most often.

The most defining generalization in the SIRP are the time synchronization constraints. Only one Branch and Cut approach for a simplified problem has been found. Nearly all heuristic approaches for the VRP with time synchronization were single-agent local search algorithms, with additional actions to account for the synchronization constraints. Two approaches can be identified. Firstly there is the move and reoptimise approach. In this approach a local search like move is done, and afterwards a new algorithm optimises the rest of the routes for the synchronization constraints. Secondly, there is the destroy and rebuild operation, where each move in the local search is defined of deleting a part of the solution and rebuilding it.

A large missing gap are the installation options. Nearly all VRP problems use a single visit per node, so even when considering one installation method very little literature was found. This was found in the research of synchronization constraints. Here, some constraints were found to model multiple visits per node, or the abstract models in the network consisted of multiple abstract nodes per location, one for each vehicle visit.

For the CRP, it can be concluded that the relocation of transition pieces is a quite unique problem as no optimisation has been found for two dimensional grid based locations. Multiple relevant problems have been studied, although the characteristics are still quite different.

For container and factory component relocation problems most research was based on either loading, unloading or pre-marshalling. No methods were discovered which integrated these parts. It is also noted that no MILP formulation was found for a sequence of batches to unload. All MILP models found considered a fixed sequence of departing items. Conversely, in train shunting operations arrival and departure was included, but this did not allow for reshuffling of objects and thus only considered assignment of storage locations.

When considering MILP formulation, most were strongly dependent on the stack based characteristics of the problems. The approach of Caserta et al. [2012] was one which considered decision variables which allowed relocation between any two positions, while blocking the disallowed locations by constraints. Therefore, it seems that the stacking characteristics are easily changed in this formulation.

With regards to heuristics it can be noted that there was less uniformity than in the vehicle routing problem, where most heuristics were a type of local search. A lot of heuristics where very problem specific algorithms. For MILP based heuristic, the rolling horizon method was used in multiple articles.

6

Component Relocation Model

The Component Relocation Problem handles the relocations of turbine components during storage in the harbour. The key characteristic is the fact that they are stored in a grid with spatial blocking constraints due to neighbouring items. The model assumes a fixed arrival and departure schedule of components, which results in the following question: how can this schedule be executed while minimizing the amount of component relocations?

A field consists of multiple storage locations, placed in a square grid. Some of these locations are loadout locations. A loadout location is a place where transfer between vessel and harbour is possible. Physically, this means that those places are located directly next to the water. The other places are called storage-only locations. If a components is to be placed from a vessel to a storage-only location or vice versa, it has to pass through a loadout place.

The crane actions can be divided into three categories. The first two categories consider the relocations from the harbour to the vessel and vice versa. These are called the loading and unloading phases respectively. The third category is called the pre-marshalling phase. In this phase, components are relocated to prepare the field layout for the loading or unloading phase.

To minimise the costs of ship rental times, project makespan has the priority. To take this into consideration for storage management, it is desired to minimise the amount of loading and unloading time for ships. This is achieved by requiring that both the loading and unloading phase are completed with as few relocations as possible. Therefore, the rule is added that during these phases there can be no other relocations besides direct loading or unloading moves. An (un)loading phase is finished after all included components are relocated to or from the vessel.

Based on approaches in literature, time is discretised in movements. This means that there is no notation of real time in the model, but only an amount of moves. This mean that crane movement distance is not considered and each relocation is assumed to take the same amount of time.

Another assumption made, is that there is no limit on the pre-marshalling phase. In reality, there is a limit on the moves in this phase, because this phase has to be finished when the next ship arrives. However, since a trip to install transition pieces might take multiple days, this limit is very high and it can be safe to assume that pre-marshalling will always be finished before the next ship arrives.

The problem is divided into three parts. Initially, only loading into ships is considered from a one dimensional storage area. This is a row where the blocking constraints are based on the left and right neighbour. After a model for this problem is given, the problem is expanded by considering a two dimensional grid instead of a one dimensional one. Subsequently, arrival and departure in batches are added, in contrast to only considering departure in a fixed sequence.

6.1. One Dimensional loading

One dimensional unloading considers the following problem: components are located in a one dimensional row, consisting of a fixed amount of storage locations. They need to be transported in a certain sequence to a vessel, but due to physical limitations they can only be moved if there is free space on either the left or the right side. This can be seen in Figure 6.1. Here, a set of 3 components is to be unloaded in ascending order. Because component 1 has no free side, component 3 is moved one location to the right. Subsequently, all component can be loaded to vessel. Therefore, it takes 3 loading moves and 1 reshuffling move to load these components.

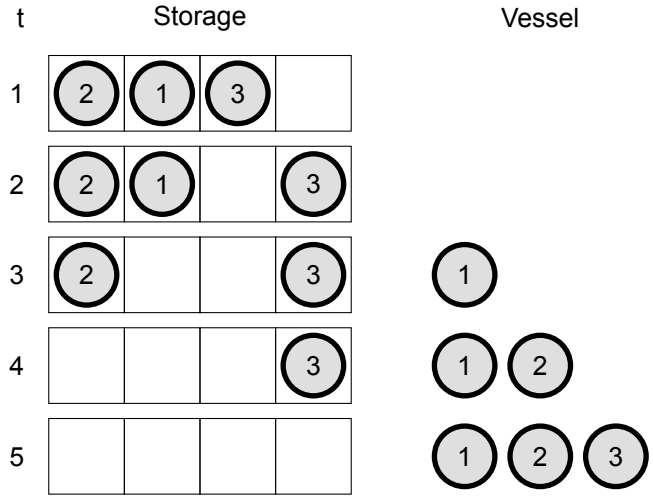


Figure 6.1: Loading of TPs to a vessel in a one dimensional row

To model this problem, decision variables based on Caserta et al. [2012] are used. Storage-only locations are not considered, and thus loading is allowed to take place from any location. The following sets are used in this problem: N contains all components. All storage locations are contained in S^s and the vessel is modelled as one location: s^d. All locations are stored in S, thus $S = \{s^d\} \cup S^s$. Additionally, the set P contains all allowed paths. Path (i, j) is an allowed path, if a component in an empty field is allowed to move from i to j. In the 1 dimensional loading problem, there is an allowed path between each pair of storage locations and a path from each storage location to s^d, but not from s_d to any other location. Furthermore, t^e is defined as $t^e = |T| + 1$. This variable can be used to get the status after the last timestep.

The binary decision variable x_{tnij} is equal to 1, if at time t, component n is moved from location i to j, and 0 otherwise. This is shown in Equations (6.3j) and (6.3k). To define these x variables, the timeset T is introduced which contains all possible times when a move can be done. Consequently, the maximum amount of steps done is $|T|$. For a shorter notation the variable y_{tni} and y_{ti} are introduced. These variables represent a sum of x variables. The binary variable y_{tni} is equal to 1 if and only if component n is positioned at location i at the beginning of time t, and y_{ti} is equal to 1 if and only if any component is positioned at location i at the beginning of timestep t.

All variables are summarised in Table 6.1

Before introducing the model with cost function and constraints, variable y is expressed as a combination of x in Equations (6.1) and (6.2). Note that for $t = 0$, y_{tni} is a constant, defined by the initial amount of components of type n at location i. Equations (6.1) and (6.2) can be interpreted as the sum of all components arriving at a certain position, minus the departing components up to time $t - 1$, plus the components at the start.

$$y_{tni} = \sum_{t'=1}^{t-1} \sum_{j \in S} x_{t'nji} - \sum_{t'=1}^{t-1} \sum_{j \in S} x_{t'nij} + y_{0ni} \forall t \in T \cup \{t^e\} \setminus \{1\}, n \in N, i \in S \tag{6.1}$$

Variable	Description
x_{tnij}	Binary decision variable 1 if and only if component n moves from i to j at time t
y_{tni}	Binary helper variable: 1 if and only if component n is located at position i at the beginning of timestep t.
T	All timesteps
N	Set of components
P	Set of allowed paths.
S	Set of All locations
S^s	Set of storage locations
s^d	vessel (loading) location

Table 6.1: Variables for 1 dimensional unloading

$$y_{ti} = \sum_{n \in N} \sum_{t'=1}^{t-1} \sum_{j \in S} x_{t'nji} - \sum_{n \in N} \sum_{t'=1}^{t-1} \sum_{j \in S} x_{t'nij} + y_{0i} \forall t \in T \cup \{t^e\} \setminus \{1\}, i \in S \tag{6.2}$$

The cost function is the amount of total moves, which is equal to the sum of all x variables. Therefore, the cost function can be expressed as Equation (6.3a). Subsequently, Equation (6.3b) imposes that only one component can be placed at a location. This equation is added for all locations except s^d, since here it is allowed to have multiple components stored.

$$\min \sum_{t \in T} \sum_{n \in N} \sum_{i \in S} \sum_{j \in S} x_{tnij} \tag{6.3a}$$

$$y_{ti} \leq 1 \forall t \in T, i \in S_s \tag{6.3b}$$

Next, the basic relocation characteristics are defined without the neighbour based blocking constraints. First, it has to be defined that a component n can only be transferred from i to j if that component is located at i. This is done in Equation (6.3c). By imposing the moves from i to any location j at time t for position n to be smaller than y_{tni}, this can only be 1 if $y_{tni} = 1$ and thus the component can move from i to somewhere else, if initially located at i.

Similarly, a component can only be moved to a location j if that location is unoccupied. This is enforced by Equation (6.3d). By adding the sum of all moves to j at time t of component n and the sum of all components at position j at time t, only one of these can be equal to one. Therefore, a move from i to j is only possible if no component is available at location j. Finally, to complete the basic relocation constraints, Equation (6.3e) is added. This limits the maximum moves per timestep to one.

$$\sum_{j \in S} x_{tnij} - y_{tni} \leq 0 \forall t \in T, n \in N, i \in S \tag{6.3c}$$

$$\sum_{i \in S} x_{tnij} + y_{tj} \leq 1 \forall t \in T, n \in N, j \in S \tag{6.3d}$$

Now that the characteristics of movements are given, the movement goals need to be specified. Since there is no relocation possible from the vessel location to any storage location due to the set of allowed paths P, the following two restrictions suffice to specify ordered unloading: The components have to move to the vessel location in order and all components have to be relocated to the vessel location. The latter one is shown in Equation (6.3f). Here, the sum of all relocation moves from the storage area to the vessel location is set equal to the amount of components.

Ordering the components to be unloaded in ascending order can be done by requiring the following: Any component n, except the first one, can only be loaded onto the vessel, if component $n-1$ is already loaded onto the vessel. This is shown in Equation (6.3g).

$$\sum_{t\in T}\sum_{n\in N}\sum_{i\in S_s} x_{tnis^d} = |N| \tag{6.3f}$$

$$y_{tns^d} - y_{t(n-1)s^d} \leq 0 \forall t \in T, n \in N \setminus \{1\} \tag{6.3g}$$

With these constraints, all components will be relocated in ascending order of N to the vessel, without any physically impossible moves. The only thing missing are the neighbour-based constraints. For a move from i to j, let's first look at the source location i. It is noted that the leftmost and rightmost locations are always accessible. For the other locations there has to be at least one free neighbour. This free neighbour requirement can be interpreted as following: A location i is considered unblocked, if the sum of all neighbours is at most 1. In Equation (6.3h), the sum of all neighbours, plus all moves from i, is set lesser or equal than 2. This results in an upper limit of 1 move in case of 1 or less neighbours, and an upper limit of 0 moves in case of two neighbours. This equation is added for every location, except the outer ones.

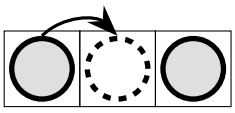


Figure 6.2: Component blocking itself

The same principle can be used to impose destination j to have maximal one neighbour. However, one thing has to be noted. When a component is repositioned to its direct neighbouring location, the target location might seem blocked, while in reality it isn't. This is illustrated in Figure 6.2. Here, the middle component has two neighbours and therefore no component can be placed here. In reality, the component will not block itself, thus the move in Figure 6.2 would be possible. Therefore, moves from a direct neighbour are excluded in Equation (6.3i). This principle is termed as the self-blocking principle, and will also be considered in blocking constraints introduced later in this thesis.

$$\sum_{n\in N}\sum_{j\in S} x_{tnij} + y_{t(i-1)} + y_{t(i+1)} \leq 2 \forall t \in T, i \in S \setminus \{1, |S|\} \tag{6.3h}$$

$$\sum_{n\in N}\sum_{i\in S\setminus\{j-1,j+1\}} x_{tnij} + y_{t(j-1)} + y_{t(j+1)} \leq 2 \forall t \in T, j \in S \setminus \{1, |S|\} \tag{6.3i}$$

The complete model is given in Equations 6.3 and is repeated below. Equations (6.3a) to (6.3g) and Equations (6.3j) to (6.3k) give a model for ordered unloading from any geometry of fixed positions to a target destination, where components can freely be swapped between any two positions as long as there is an allowed path. Although the solution for this model will always be trivial, it provides a good basis for more complicated models. By adding Equations (6.3h) and (6.3i), the neighbour-based relocation constraints are added. In the next section, these will be replaced by two dimensional relocation constraints.

$$\min \sum_{t\in T}\sum_{n\in N}\sum_{i\in S}\sum_{j\in S} x_{tnij} \tag{6.3a}$$

subject to

$$y_{ti} \leq 1 \forall t \in T, i \in S_s \tag{6.3b}$$

$$\sum_{j\in S} x_{tnij} - y_{tni} \leq 0 \forall t \in T, n \in N, i \in S \tag{6.3c}$$

$$\sum_{i\in S} x_{tnij} + y_{tj} \leq 1 \forall t \in T, n \in N, j \in S \tag{6.3d}$$

$$\sum_{n\in N}\sum_{i\in S}\sum_{j\in S} x_{tnij} \leq 1 \forall t \in T \quad (6.3e)$$

$$\sum_{t\in T}\sum_{n\in N}\sum_{i\in S_s} x_{tnis^d} = |N| \quad (6.3f)$$

$$y_{tns^d} - y_{t(n-1)s^d} \leq 0 \forall t \in T, n \in N \setminus \{1\} \quad (6.3g)$$

$$\sum_{n\in N}\sum_{j\in S} x_{tnij} + y_{t(i-1)} + y_{t(i+1)} \leq 2 \forall t \in T, i \in S \setminus \{1, |S|\} \quad (6.3h)$$

$$\sum_{n\in N}\sum_{i\in S\setminus\{j-1,j+1\}} x_{tnij} + y_{t(j-1)} + y_{t(j+1)} \leq 2 \forall t \in T, j \in S \setminus \{1, |S|\} \quad (6.3i)$$

$$x_{tnij} \in \{0,1\} \forall t \in T, n \in N, (i,j) \in P \quad (6.3j)$$

$$x_{tnij} = 0 \forall t \in T, n \in N, (i,j) \notin P \quad (6.3k)$$

6.2. Two dimensional relocation constraints

The model is now expanded to contain two dimensional blocking constraints. As shown earlier, the field consists of a grid of up to two layers with a crane on a rails next to it as shown in Figure 2.9. In the remainder of this thesis, the field will always be described from the position of the crane. The front row is the row which is closest to the crane, and behind that is the back row. The back row is positioned at the bottom in every figure, and the front row at the top.

Recall from Chapter 2 that a component is blocked if it has no free side or when there is a component in front of it. Having a free side means, for the locations in the front row, that on either the left or right side, there is a free space. The same holds for the locations in the back row, but additionally there also must be a free space in front of this free side neighbour.

This results in different blocking constraints for different locations. For components in the front row, only the left or right neighbour has to be unoccupied. For components in the back row, all five surrounding components have to be considered. To implement this in the model, the term relevant side neighbour is introduced. The relevant side neighbours are the neighbours on both rows on both sides, which have to be considered in deciding the blocked-state of a location. This is shown in Figure 6.3. Note that only side neighbours, thus neighbours in the adjacent columns, are considered here. The reason for this is that the blocking constraints due to components in front of other components are added separately.

Locations with side neighbours at only one side are never blocked by the side neighbour based blocking constraints, since there is always a free side. The same holds for locations at the front row with only one side neighbour in the same row. These locations therefore have zero side neighbours to consider. Next to this, all other locations at the front row only have the left and right side neighbours to consider, and therefore these positions have two relevant side neighbours. Furthermore, ,it can be seen in Figure 6.3 that there are three back row locations with one neighbour on one side and two on the other side, resulting in three relevant side neighbours. The middle back component has all 4 relevant side neighbours to take into consideration.

This information is stored in the sets M_a^s and R_{bi}^s. M_a^s contains all locations with exactly a relevant side neighbours. R_{bi}^s with $b \in \{1,2\}$ contains the two sets (left and right) of relevant side neighbours of location i. In case of three relevant neighbours, R_{1i}^s represents the neighbour-set with length 1, and R_{2i}^s represents the set of two neighbours. Similarly, the locations with a storage location in front of it are contained in M^h, and for each of these locations R_i^h defines the

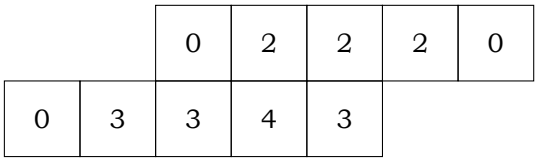


Figure 6.3: Relevant side neighbours per location

location above i. Besides these, the set of all relevant neighbours of location i is defined under R_i^s. R_{bim}^s denotes the mth element in R_{bi}^s. If there is more than 1 element in R_{bi}^s, then $m = 1$ denotes the back row.

These sets are summarised in Table 6.2.

Set	Description
M_a^s	Set of locations with exactly a relevant side neighbours
M^h	Set of locations with one location in front it
R_{1i}^s	Smallest set of side neighbours (left or right) of location i
R_{2i}^s	largest set of side neighbours (left or right) of location i
R_i^h	Defines the location above i.
R_i^s	All relevant side neighbours of i
R_{aim}^s	The mth location in R_{ai}^s, with the highest m being the most front one

Table 6.2: Sets to describe neighbouring components

With these sets, the relocation constraints of the two-dimensional problem can be defined. As stated before, the relocation constraints of side neighbour based blocking and front-neighbour based blocking are added separately. This is done because there is an and-relationship between these two: One side must be free and the top must be free. In linear programming, and-relationships are simple and can be added by separate constraints. The top-neighbour constraints are added first. These constraints are shown in Equations (6.7a) and (6.7b). For a move from i to j, i is required to be unblocked in Equation (6.7a) and j is required to be unblocked in Equation (6.7b). In these equations, either the total amount of relocations to or from a location, or the total amount of components present at the front position (second part), can be 1, but not both. In Equation (6.7b) relocations from the front position to the back position are excluded due to the self-blocking principle.

$$\sum_{n\in N}\sum_{j\in S} x_{tnij} + y_{tR_i^h} \leq 1 \forall i \in M^h, t \in T \tag{6.7a}$$

$$\sum_{n\in N}\sum_{i\in S\setminus\{R_j^h\}} x_{tnij} + y_{tR_j^h} \leq 1 \forall j \in M^h, t \in T \tag{6.7b}$$

This remains the side neighbouring based constraints. There are three types here, those with two, three and four relevant side neighbours. First, the locations with two relevant side neighbours are added. These constraints are the same as in the one-dimensional problem and are written based on the introduced sets in Equations (6.7c) and (6.7d).

$$\sum_{n\in N}\sum_{j\in S} x_{tnij} + y_{tR_{1i1}^s} + y_{tR_{2i1}^s} \leq 2 \forall t \in T, i \in M_2^s \tag{6.7c}$$

$$\sum_{n\in N}\sum_{i\in S\setminus R_i^s} x_{tnij} + y_{tR_{1i1}^s} + y_{tR_{2i1}^s} \leq 2 \forall t \in T, j \in M_2^s \tag{6.7d}$$

For the back rows, the constraints are less simple. The locations with four relevant side neighbours are discussed first. The side based constraints can be described as follows: A location i is unblocked based on side neighbours constraints, if either the front and back row locations on the left, or the front and back row locations on the right, are unoccupied. In Equations (6.7c) and (6.7d) the principle was used that out of two positions, at least one has to be unoccupied. If this principle would be used for the four neighbour constraints, thus requiring two out of four locations to be unoccupied, the back middle component in Figure 6.5 would not be considered blocked.

Therefore, consider location i. The variables L^f,L^b,R^f and R^b, respectively denote the left front, left back, right front and right back locations as shown in Figure 6.4. According to

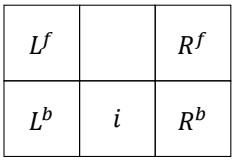


Figure 6.4: Notation during explanation of side constraints

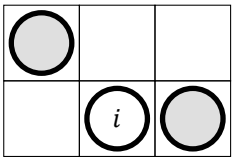


Figure 6.5: Layout where i would be considered unblocked if a 2 out of 4 principle would be used

Lemma 6.2.1, relocation from i can be blocked if there is not one free neighbour side, by adding Equations (6.7e) and (6.7f).

$$2y_{tR^s_{1i1}} + y_{tR^s_{2i1}} + y_{tR^s_{2i2}} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M^s_4 \tag{6.7e}$$

$$2y_{tnR^s_{1i2}} + y_{tR^s_{2i1}} + y_{tR^s_{2i2}} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M^4_s \tag{6.7f}$$

Lemma 6.2.1. *Let x'_i be 1 if and only if a component is relocated from a location i with 4 relevant side neighbours, and let L_u,L_d,R_u and R_d denote the binaries equal to one if and only if respectively the left upper, left bottom, right upper or right bottom location is occupied. Then Equation* (6.4) *and Equation* (6.5) *block relocations from i if and only if there is not one free side.*

$$L^b + L^f + 2R^f + 2x'_i \leq 4 \tag{6.4}$$

$$L^b + L^f + 2R^b + 2x'_i \leq 4 \tag{6.5}$$

Proof. If at least one component is placed in the left column, either L^f, L^b, or both are equal to 1. Then Equation (6.4) becomes either $2R^f + 2x'_i \leq 3$ or $2R^f + 2x'_i \leq 2$, and Equation (6.5) will become either $2R^b + 2x'_i \leq 3$ or $2R^b + 2x'_i \leq 2$. If also one component is placed in the right column, either R^f, R^b, or both are equal to 1. If $R^f = 1$, Equation (6.4) becomes $2x'_i \leq 0$ or $2x'_i \leq 1$. If $R^b = 1$, Equation (6.5) becomes $2x'_i \leq 0$ or $2x'_i \leq 1$.

Conversely, if the whole left column is unoccupied, Equation (6.4) and Equation (6.5) both become $2 * R^f + 2x'_i \leq 4$ and $2 * R^b + 2x'_i \leq 4$, respectively. Therefore, the constraints are never violated in this case.

If the right column is unoccupied, both Equations (6.4) and (6.5) become $L^b + L^f + 2x'_i \leq 4$, which also can never be violated. □

A similar approach can be used for relocations from a location, although the self-blocking principle has to be taken into account. For this purpose, let's consider constraints in the form of Equation (6.5), with x'_i denoting the moves to i from location $j \notin \{L^f, L^b, R^b\}$. Note that relocations from L^b and R^b do not have to be considered, since components at these location can always be moved to i if there is no component in front of them and are always blocked when there is. Therefore, blocking of these relocations are already imposed due to the front-blocking constraints. Furthermore, by Lemma 6.2.2, the self blocking principle is taken care of for the right upper position.

Therefore, adding a symmetrical equation for the left upper position, and an equation in the form of Equation (6.5), completes the blocking constraints without violating the self-blocking principle for all neighbouring components. This gives in Equations (6.7g) and (6.7h).

It remains to add blocking constraints for all non-neighbouring locations. Since an equation in the form of Equation (6.4) was already added, by Lemma 6.2.1, this can be completed by adding an equation in the form Equation (6.5), where x'_i denoting the moves to i from any non-neighbouring location. Therefore, Equation (6.7i) is added. .

$$2y_{tR^s_{1i1}} + y_{tR^s_{2i1}} + y_{tR^s_{2i2}} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M^s_4, S' = S \setminus \{R^s_{1i1}, R^s_{2i1}, R^s_{2i2}\} \tag{6.7g}$$

$$2y_{tnR^s_{1i2}} + y_{tnR^s_{2i1}} + y_{tnR^s_{2i2}} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M^s_4, S' = S \setminus \{R^s_{1i2}, R^s_{2i1}, R^s_{2i2}\} \tag{6.7h}$$

$$2y_{tR^s_{2i1}} + y_{tnR^s_{1i1}} + y_{tR^s_{1i2}} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M^s_4, S' = S \setminus \{R^s_{2i1}, R^s_{1i1}, R^s_{1i2}\} \tag{6.7i}$$

Lemma 6.2.2. *Let x'_i be 1 if and only if a component is placed to a location i with 4 relevant side neighbours from any non-neighbouring place or the right upper neighbour, and let L^f,L^b,R^f and R^b denote the binaries equal to one if and only if the left front, left back, right front and right back location is occupied, respectively. Then Equation* (6.5) *blocks relocations from the right upper location to location i if the left side is not completely unoccupied and the right bottom location is occupied*

Proof. If the left side is not completely unoccupied and the right bottom side is occupied, Equation (6.5) becomes $\{0, 1\} + 2x'_i \leq 1$. Therefore, a move from the right top location to location i is blocked. □

For the three side neighbours, a single equation in the form of Equation (6.4) suffices for defining the blocking constraints on relocation from a location i. This gives Equation (6.7j). Furthermore, if Equation (6.7k) is added, blocking requirements for all relocations to location i from any location except the side neighbours are added.

Furthermore, relocations from R^s_{1i1} are always allowed, and relocations from the back row in R^s_{2i} are allowed if and only if there is no component in front if it. Therefore, only the front component of R^s_{2i} has to be considered. This component can be relocated to i if and only if R^s_{1i1} is not occupied, so by adding Equation (6.7l) this constraint is added.

$$2y_{tR^s_{1i1}} + y_{tR^s_{2i1}} + y_{tR^s_{2i2}} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M^s_3 \tag{6.7j}$$

$$2y_{tR^s_{1j1}} + y_{tR^s_{2j1}} + y_{tR^s_{2j2}} + 2\sum_{n\in N}\sum_{i\in S\setminus R^s_j} x_{tnij} \leq 4 \forall t \in T, j \in M^s_3 \tag{6.7k}$$

$$y_{tR^s_{1j1}} + \sum_{n\in N} x_{tnR^s_{2j2}j} \leq 1 \forall t \in T, j \in M^s_3 \tag{6.7l}$$

In the two-dimensional grid, loadout locations S^{sl} have been introduced. These are locations from which loading and unloading to the ship is possible. This can be done by removing the paths from the storage-only locations S^{st} to the vessel loading location, as shown in Equation (6.6). With this, the two dimensional loading model can be defined by combining Equations 6.7, Equations (6.3a) to (6.3g), Equation (6.3j) and Equation (6.3k). The new equations are displayed again in Equations 6.7

$$P = \{(i,j)|i \in S_s, j \in S^s\} \cap \{(i,j)|i \in S^l_s, j \in S^s\} \tag{6.6}$$

$$\sum_{n\in N}\sum_{j\in S} x_{tnij} + y_{tR_i^h} \leq 1 \forall i \in M^h, t \in T \tag{6.7a}$$

$$\sum_{n\in N}\sum_{i\in S\setminus R_j^h} x_{tnij} + y_{tR_j^h} \leq 1 \forall j \in M^h, t \in T \tag{6.7b}$$

$$\sum_{n\in N}\sum_{i\in S} x_{tnij} + y_{tR_{1i1}^s} + y_{tR_{2i1}^s} \leq 2 \forall t \in T, i \in M_2^s \tag{6.7c}$$

$$\sum_{n\in N}\sum_{i\in S\setminus R_i^s} x_{tnij} + y_{tR_{1i1}^s} + y_{tR_{2i1}^s} \leq 2 \forall t \in T, j \in M_2^s \tag{6.7d}$$

$$2y_{tR_{1i1}^s} + y_{tR_{2i1}^s} + y_{tR_{2i2}^s} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M_4^s \tag{6.7e}$$

$$2y_{tnR_{1i2}^s} + y_{tR_{2i1}^s} + y_{tR_{2i2}^s} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M_4^s \tag{6.7f}$$

$$2y_{tR_{1i1}^s} + y_{tR_{2i1}^s} + y_{tR_{2i2}^s} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M_4^s, S' = S \setminus \{R_{1i1}^s, R_{2i1}^s, R_{2i2}^s\} \tag{6.7g}$$

$$2y_{tnR_{1i2}^s} + y_{tnR_{2i1}^s} + y_{tnR_{2i2}^s} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M_4^s, S' = S \setminus \{R_{1i2}^s, R_{2i1}^s, R_{2i2}^s\} \tag{6.7h}$$

$$2y_{tR_{2i1}^s} + y_{tnR_{1i1}^s} + y_{tR_{1i2}^s} + 2\sum_{n\in N}\sum_{i\in S'} x_{tnij} \leq 4 \forall t \in T, j \in M_4^s, S' = S \setminus \{R_{2i1}^s, R_{1i1}^s, R_{1i2}^s\} \tag{6.7i}$$

$$2y_{tR_{1i1}^s} + y_{tR_{2i1}^s} + y_{tR_{2i2}^s} + 2\sum_{n\in N}\sum_{j\in S} x_{tnij} \leq 4 \forall t \in T, i \in M_3^s \tag{6.7j}$$

$$2y_{tR_{1j1}^s} + y_{tR_{2j1}^s} + y_{tR_{2j2}^s} + 2\sum_{n\in N}\sum_{i\in S\setminus R_j^s} x_{tnij} \leq 4 \forall t \in T, j \in M_3^s \tag{6.7k}$$

$$y_{tR_{1j1}^s} + \sum_{n\in N} x_{tnR_{2j2}^s j} \leq 1 \forall t \in T, j \in M_3^s \tag{6.7l}$$

6.3. Batches

So far, only loading in a certain order, with an initial field of components, was considered. During an installation project, the components will arrive and depart in batches. For this purpose, one more location is added: The arrival location, located at $i = 0$. All components start in this position. Paths are added from this location, to the loadout area. These are called arrival paths, as shown in Equation (6.8). Together with the field paths (Equation (6.9)) and departure paths (Equation (6.10)), the total set of paths is defined as Equation (6.11).

$$P^a = \{(0, j) | j \in S^{sl}\} \tag{6.8}$$

$$P^f = \{(i, j) | i \in S^s, j \in S^s\} \tag{6.9}$$

$$P^d = \{(i, s^d) | i \in S^{sl}\} \tag{6.10}$$

$$P = P^a \cup P^f \cup P^d \tag{6.11}$$

Ships, referred to as batches, arrive in order of $B = \{1, .., b_{max}\}$. Each batch b has a type bt_b(arriving or departing), and P^{bt_b} denote the corresponding paths (P^a or P^d). Furthermore, a batch b includes components N_b^B and is called completed if all components N_b^B if all components have crossed a path in P^{bt_b}. To fulfil the schedule of arriving and departing ships all batches have to be completed in order.

Variable	Description
B	Batches
P^a	Arrival paths
P^d	Departure paths
bt_b	Type of batch b
N_b^B	Components in batch b
P^f	Field paths
$P^b t_b$	Paths corresponding to bt_b

Table 6.3: Batch variables

To complete an arriving batch b, all components have to be moved to the loadout area. This is expressed in Equation (6.12). This equation sets the sum of all components of batch b which are relocated over the arrival paths P^a, equal to amount of components in batch b. For completing a departing batch the same can be done but for paths P^d.

$$\sum_{t \in T} \sum_{n \in N^b} \sum_{(i,j) \in P^a} x_{tnij} = |N_b^B| \tag{6.12}$$

To require ordered completion of these batches, constraints are added for each pair of two subsequent batches b and d (thus $b + 1 = d$). The aim of these constraints is to disallow any movement of a component in N_d^B over P^{bt_d}, before all moves of components N_b^B over P^{bt_b} are finished. This is done in Equation (6.13a). The first part of this equation exists of one (un)loading move from $b + 1$. If a move is done, the second part has to be equal to $|N_b^B|$. This means that all (un)loading moves from batch b have to be done.

Finally, it is required that during the loading or unloading operation no other moves are done. If this is not included, other relocations might happen during (un)loading which might cause delay for the ships. The constraint added for this is Equation (6.13b). This constraint is added for each batch b, timestep t except t_{max} and each component n in N^b. The terms in this equation are from left to right: If n is (un)loaded at time t, the total (un)loading flow at $t + 1$ and the total amount of items of N_b unloaded until t.

If the first term is zero, this equation is always valid. This means that the equation only matters if n is (un)loaded at time t, in which case the first term is equal to $|N_b^B|$. The second term can take on either 0 or $|N_b^B|$. If it is zero, then the third term has to be equal to $|N_b^B|$ to satisfy the inequality. This means that if one item is (un)loaded from N_b^B at time t, and in the next timestep nothing is unloaded, then $|N_b^B|$ items have to be (un)loaded at the end of time t. However, if the second term is equal to $|N_b^B|$, the inequality is always valid. This results in the rule that if one component of the batch is (un)loaded, either it has to be the last of the batch, or in the next timestep another item in batch b has to be (un)loaded. This satisfies the direct unloading requirement.

Therefore, the complete CRP model is defined in Equations 6.7, Equations (6.3a) to (6.3g), Equation (6.3j), Equation (6.3k), Equation (6.13).

$$|N_b^B| \sum_{(i,j) \in P^{bt_{b+1}}} x_{tnij} - \sum_{t'=1}^{t} \sum_{n' \in N^b} \sum_{(i,j) \in P^{bt_b}} x_{t'n'ij} \leq 0 \forall t \in T, b \in B \setminus \{b_{max}\}, n \in N_{b+1}^B \tag{6.13a}$$

$$|N^b| \sum_{(i,j) \in P^{bt_b}} x_{tnij} - |N_b| \sum_{n' \in N^b} \sum_{(i,j) \in P^{bt_b}} x_{(t+1)n'ij} - \sum_{n' \in N_b} \sum_{t'=1}^{t} \sum_{(i,j) \in P^{bt_b}} x_{t'n'ij} \leq 0 \forall t \in T \setminus \{t_{max}\}, b \in B, n \in N_b^B \tag{6.13b}$$

6.4. Discussion

Whenever a component transfers between the storage-only area and a vessel, it has to go through the loadout area. This was done by defining the set of paths P. Although this method is very convenient and adaptable, it does cause an extra step. Given a transfer of a component between the vessel and the storage-only area, it is a trivial task to define the intermediate loadout position as long as there is space available. To solve this trivial task two extra steps are needed per component, thus increasing the amount of variables. Therefore, it might be interesting to explore the possibilities of a direct path between the vessel and storage-only area, with increased cost to account for the intermediate loadout step without directly modelling it. This would need extra constraints however, which would define of these virtual paths can be used, based on the current loadout occupation.

Furthermore, it is interesting, and important, to gain more knowledge what the possibilities are of linear inequalities to model component blocking. This is very problem specific, and although moderately complicated blocking constraints have been modelled here not much can be said about what kind of constraints are impossible. During the Walney Installation project the movement of monopiles also included blocking constraints. Although not included in the scope of this thesis, it was shown in Appendix C how these blocking constraints could be formulated by constraints compatible with the current crane optimisation model.

6.5. Conclusion

The basic model without the spatial relocation constraints gives a very general model for loading and unloading problems. This can also be of use for other components during offshore wind farm installations, or for relocation problems in general.

The path based approach makes this model suitable for other problem where flow between locations is not always possible. This might for example be in a storage field existing of different parts, where transfer between parts can only happen at certain locations. This can easily be modelled by simple deleting or adding different paths.

The model supports both initial components in the field as arriving components. All reshuffling will be executed in between (un)loading phases due to Equation (6.13b). The relocation constraints have been added for the Walney transition pieces. For different geometrical constraints, most of the current model can be used. Only the blocking constraints have to be reformulated.

7

Crane optimisation exact

To find an optimal solution for the model introduced in Chapter 6, it was implemented in the MILP solver of CPLEX Optimization Studio. To improve the performance of this solver, cutting planes were introduced. Cutting planes are additional constraints, added with the goal of improving the formulation, as discussed in Chapter 4. Besides this, some changes in implementation have been made for the variables, to both reduce the memory needed to store the problem, and to reduce the time needed for solving. These changes in variables are discussed first, after which the cutting planes are presented.

7.1. Variables

Two changes have been made to the variables. First, the paths have been reduced, and therefore the amount of x variables. Secondly, the variable y was introduced to decrease the non-zeros in the constraint matrix. This had a positive effect on the solver efficiency and also on the memory used.

Instead of a fixed set of paths, the paths were modified to be component and timestep specific. The path indices were changed from (i,j) to (t,n,i,j). If a path does not exist, the corresponding x variable is fixed at zero. This makes it possible to reduce the amount of variables in the problem. For each arriving batch b, the earliest possible arriving time is defined as tb_b^{min}. A lower bound on this is the amount of components in the preceding batches. Similarly, tb_b^{max} denotes the latest possible departure time of a departing batch. Furthermore, b_n^d and b_n^a are introduces as, respectively, the departing and arrival batches containing component n.

Variable	Description
tb_b^{min}	Earliest arrival time for any arriving batch b
tb_b^{max}	Latest departure time for any departing batch b
b_n^a	Arrival batch of component n
b_n^d	Departing batch of component n

Table 7.1: Variables for path definition

The paths can now be defined. In Equation (7.1), all paths from the arrival location to the loadout are defined. The latest time of these paths is subtracted by the length of the departing batch, since they already need to be moved to the field at that time. The opposite is done in Equation (7.2). Here, the paths only exist $|N^{b_n^a}|$ timesteps after the earliest possible arrival time. For the intrafield paths, P^f this is combined, as shown in Equation (7.3)

$$P^a = \{(t,n,0,j) | j \in S^{sl}, t \in T, n \in N, t \geq tb_{b_i^a}^{min}, t \leq tb_{b_i^d}^{max} - |N_{b_i^d}|\} \tag{7.1}$$

$$P^d = \{(t,n,i,s_d) | i \in S^{sl}, t \in T, n \in N, t \geq tb_{b_n^a}^{min} + |N_{b_n^a}|, t \leq tb_{b_n^d}^{max}\} \tag{7.2}$$

$$P^f = \{(t,n,i,j) | i \in S^f, j \in S^f, t \in T, n \in N, t \geq tb_{b_n^a}^{min} + |N_{b_n^a}|, t \leq tb_{b_n^d}^{max} - |N_{b_n^d}|\} \tag{7.3}$$

7.2. Cutting Planes

Cutting planes are additional equations which do not add new characteristics to the model, but remove a part of the search space to improve solver performance. Three types of these equations where added. First, a lower bound on the amount of moves is calculated. Secondly, time symmetry is removed and thirdly, a type of move which is never included in an optimal solution is forbidden.

Both lower bounds are based on a simulation of the arriving and departing batches, without considering the blocking constraints. A lower bound lb_x is calculated based on the forced moves from loadout. This algorithm can be described as following. For each batch, the length of that batch is added to lb_x. If it is an arriving batch and there is not enough space on the loadout area, the extra space needed is also added to lb_x. Simultaneously, a set of all possible components on loadout N^l is kept, by adding all arriving components per batch. When the loadout area is completely replaced in one arrival batch, or cleared in a departing batch, N^l is reset. When a departing component is not in N^l, it means that it must be in the storage only-area. Therefore, if this component departs, lb_x is incremented by one to represent the move from the storage-only location to the loadout location. This lb_x is than used as a lower bound for x, as shown in Equation (7.4).

$$\sum_{t \in T} \sum_{n \in N} \sum_{i \in S} \sum_{j \in S} x_{tnij} \geq lb_x \tag{7.4}$$

This can be done similarly for the amount of moves, plus the moves left. This is applicable when considering partial loadout, thus with an initial and/or a remaining amount of components in the storage area. For this, no list of possible components at the loadout area is needed. The lower bound is defined as lb_{xy}. This is initialised as the amount of components in the storage-only area. For these components, it holds that either they have to be replaced, causing x to increase with one, or they will end at the storage-only area, causing y to increase by one. When a new batch b arrives and extra space is needed at the loadout area, lb_{xy} is increased by $2 * |N_b|$. One for moving to the storage-only area, and one for either staying there or moving away. This lower bound is then added as shown in Equation (7.5)

$$\sum_{t \in T} \sum_{n \in N} \sum_{i \in S} \sum_{j \in S} x_{tnij} + \sum_{n \in N} \sum_{i \in S_s} y_{t^e i} \geq lb_{xy} \tag{7.5}$$

The next cutting plane removes time-symmetry. Often, there are more timesteps then moves needed. Give a sequence of moves, these moves can be fitted in T in multiple ways. For example, consider the moves sequence (a,b) with $|T| = 3$ and let _ denote a timestep without any movement. The solutions $(a,b,\_)$,$(a,\_,b)$ and $(\_,a,b)$ are then identical, but still represent three different solutions in the solver. These solutions are called symmetrical. To prevent exploring these symmetrical solutions, Equation (7.6) is added. The terms in this equation represent the amount of moves on $t+1$ and t. By setting the difference smaller than zero, it holds that if there is a move at $t+1$, there has to be a move at t. This pushes all moves to the first part of T, therefore removing the symmetrical solutions.

$$\sum_{n \in N} \sum_{(i,j) \in P} x_{(t+1)nij} - \sum_{n \in N} \sum_{(i,j) \in P} x_{tnij} \leq 0 \forall t \in T \setminus \{t_{max}\} \tag{7.6}$$

Additionally, one other cutting plane is added. Consider the move i to j of component n where j is a storage-only location. If n is then relocated again in the next timestep to location

k, it holds that n could also have been located directly from i to k. Since the latter costs one move less, the former combination of moves would never happen in an optimal solution. This is added in Equation (7.7). Here the sum of both previously described moves is set to maximal one. Therefore, only on of these moves can occur.

$$\sum_{n \in N} \sum_{i \in S} x_{tnij} + \sum_{n \in N} \sum_{k \in S} x_{(t+1)tnjk} \leq 1 \forall t \in T \setminus \{t_{max}\}, i \in S_s \tag{7.7}$$

By reducing the paths, the amount of variables was reduced significantly. The change of y from an expression of x to a separate variable increased the amount of constraints and variables, but reduced the total amount of non-zeros in the constraint matrix and this trade-off turned out to be worth it by performing small tests. Furthermore, the cutting planes resulted in a reduction of the searchspace and therefore an improvement of solver performance. However, not unexpected, the component relocation problem for Walney is too large to solve to optimality. Therefore, a heuristic is developed, as will be discussed in Chapter 8.

8

Rolling Horizon Algorithm

The MILP model for the CRP gives an exact definition of the problem and can solve smaller instances to optimality. For larger instances however, no solution will be found. For this reason a heuristic method is developed.

This heuristic method is a rolling horizon heuristic, as used by Kamrad et al. [2012]. The idea of this method is to divide the whole problems in subproblems. The first subproblem is solved from the project starting time, up to a certain later time called the planning horizon. The subsequent subproblem uses moves from the preceding one to define it's initial component layout, and thus slightly shifts the timewindow of the subproblem. By combining the moves in all subproblems, a solution to the complete problem can be found.

For more clarification, the full algorithm will be given in the next section. Subsequently, the reason for selecting this heuristic will be presented. After that, more implementation details for the rolling horizon algorithm will be given. This includes modifying the MILP model and creating an algorithm to solve the subproblem.

8.1. Algorithm

The rolling horizon algorithm is given in Algorithm 8. It is initialised with a batch-horizon, a batch index set initially at 1 and an initial field. The algorithm then iteratively solves parts of the complete problem until all batches in B are processed.

Algorithm 8 Rolling horizon algorithm

H = Batch horizon
B = All batches
b = 1 ▷*B*atch index
M = {} ▷*S*tored moves
F = Initial Field
repeat
    all_moves = solve(F,b,$b+H$)
    move_subset = select_moves(all_moves)
    M = M∪move_subset
    F = update_field(F,moves_subset)
    $b = b + 1$
until $b + H = |B|$

The first step in the rolling horizon loop is solving the subproblem. The first subproblem is defined as finding an optimal solution to complete the first H batches. This will be done by solving a modified MILP model, as will be introduced later. Solving the subproblem gives all_moves. This is the optimal solution for (un)loading batches b up to $b + H$.

From these moves, only the (un)loading moves for the first batch will be selected. These are, for example, the moves to place the components of the first batch to the loadout area,

in case of arriving components. These selected moves are then stored, becoming part of the final solution. An exception for this rule is the final iteration, in which case all moves are selected and stored.

Finally, the field is updated by executing these selected moves and the batch index is incremented by one. This loop is then repeated, until the final batch is included in the subproblem. A visualization of this process is shown in Figure 8.1.

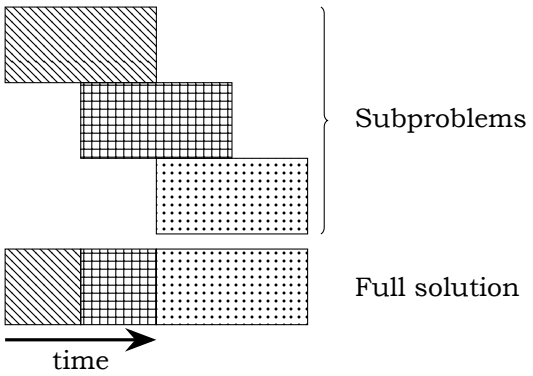


Figure 8.1: Rolling horizon illustration for three subproblems

8.2. Choice heuristics

The choice for the rolling horizon is based on literature research and problem characteristics. First of all, good results were found using this algorithm by Wan et al. [2009], who compared the rolling horizon algorithm against multiple other heuristics. It was also used by Kamrad et al. [2012], who showed its applicability on a slightly different problem, namely coil stacking.

This diverse applicability is one of the most important reasons for using the rolling horizon heuristic. Most container-relocation algorithms found in literature were problem specific algorithms, with instructions based on the goal and structure of the problem. The rolling horizon can be adapted to different problems by modifying the MILP model. Different geometrical requirements can easily be added by modifying the relocation constraints. It is also possible to relax constraints to get a lower bound. With this, it is for example possible to see what the effect would be if only the component just above a location was required to be free, instead of the side neighbours as well.

Besides this, it is also important to look at what is not done by previous authors. So far, no authors considered both loading and unloading. The characteristics of the rolling horizon seem to be favourable for adding this, because it divides the problem in smaller parts, with the possibility of separating the loading and unloading part.

Another motivation for the rolling horizon is found in the nature of installation projects. During the installation, circumstances will change and there is a need for an algorithm which can be ran during the execution of the project, to process these changes and give a new optimal schedule. For this, it might not be necessary to get the complete schedule. At the moment of arrival of a ship with components, the crane operator is mostly interesting in knowing where to place those components, and much less where the components arriving two weeks later should be placed.

The rolling horizon divides the problem into smaller subproblems. Therefore, it is possible to run the algorithm partly, for example just for the first subproblem. This would answer the question what should be done with the current components without running the complete algorithm.

In conclusion, the rolling horizon algorithm showed good results in the container relocation problem, is very adaptable and the algorithm seems promising to implement loading and unloading. Additionally, it can give partial solutions which include the moves up to a certain point in time, which is useful during installation projects. In the rest of this chapter, further implementation of the algorithm will be presented.

8.3. Subproblem MILP

The subproblem, iteratively solved during the rolling horizon algorithm, is based on the earlier given model of the crane optimisation model. This formulation already supported components in an initial field, but was modelled to load all components. This is not requested

in the subproblem, as only a subset of items (possibly zero) based on the selected batches should loaded.

By only loading a subset of all components, the final configuration will therefore contain the components left. When optimizing such a problem solely to crane movements, the final state of components is not taken into account. This might create solutions where the selected batches are (un)loaded with minimal movements, but due to the field layout at the end it might force future steps to require more moves.

To minimise this risk, some evaluation will be needed to determine the quality of the layout of components at the end of the subproblem. Three principles will be used for this: firstly the blocking principle will be introduced, which determines if neighbours of a component will need to be relocated in the future. Secondly, the principle of unusable space will be described. This will take into account positioning of components causing unusable space. Finally, a measure for clustering is added, which rewards components being placed close to each other.

8.3.1. Partial unloading

First, partial unloading will be added. In a solution for the complete problem, the final condition was that all components were loaded onto the ship. This is not the case in the subproblem. Let the batches to be (un)loaded in the subproblem be defined as B'. Furthermore, let the batch after B', if it exists, be defined as b'^f. If b'^f is an arriving batch, the ending condition is defined as follows: The subproblem is completed if all batches in B' are completed and there is at least $|N^B_{b'^f}|$ (amount of components in batch b'^f) unoccupied space in the loadout area.

Additionally, if b'^f is a departing batch, the final condition requires all components of $N_{b'^f}$ to be placed at the loadout area. If b'^f does not exists, it means that the final batch is included in the subproblem. If this is the case, the problem can simply require all component to be unloaded.

In the subproblem, Equations (6.13a) and (6.13b) for uninterrupted loadout and ordered loadout of batches, respectively, can be used by simple replacing B by B'. However, Equation (6.3f) has to be modified. Instead of requiring all components to move to the departure location s^d, only the components of the departing batches have to. Additionally, an equation has to be added to force all arriving components to the loadout area. The sets N'^a and N'^d are introduced to represent all arriving and all departing components, respectively, as given in Equations (8.1) and (8.2).

$$N'_a = \{n : n \in N^B_b, b \in B' \ , c_b = 1\} \tag{8.1}$$

$$N'_d = \{n : n \in N^B_b, b \in B' \ , c_b = 2\} \tag{8.2}$$

With these sets, the following constraints are introduced to the problem. Equation (8.3a) requires all arriving components to be moved to the loadout area, in a similar fashion of Equation (6.3f). Additionally, Equation (8.3b) requires all departing components to be loaded onto the vessel. These two equations replace Equation (8.3b). Furthermore, an inequality has to be added to prepare the loadout area for the next step b'^f. If b'^f is an arriving batch, it holds that at the final time of the subproblem, there must be enough space to unload all the components of b'^f.

This is done in Equation (8.3c). Here, the total amount of components occupying a loadout place at the end of the problem must be smaller than the total loadout space, minus the amount of components in batch b'^f. In Equation (8.3d), the total amount of components from b'^f in the loadout space at the final time is set equal to the amount of components in b'^f, therefore requiring all components needed to be on the loadout area after finishing.

Variable	Description
B'	All batches arriving and departing during subproblem
b'^f	Batch after last batch of B'
N'^a	All components in arriving batches in subproblem
N'^d	All components in departing batches in subproblem

Table 8.1: Variables used in Equations 8.3

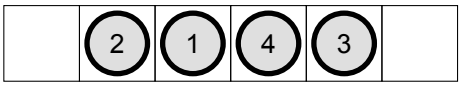


Figure 8.2: Illustration of blocked components

$$\sum_{t\in T}\sum_{n\in N'^a}\sum_{j\in S_s} x_{tn0j} = |N'^a| \tag{8.3a}$$

$$\sum_{t\in T}\sum_{n\in N'^d}\sum_{i\in S_s} x_{tnis_d} = |N'^d| \tag{8.3b}$$

$$\sum_{n\in N}\sum_{i\in S^{sl}} y_{t^e ni} \leq |S_l| - |N_{b'^f}| \tag{8.3c}$$

$$\sum_{n\in N_{b'^f}}\sum_{i\in S^{sl}} y_{t^e ni} = |N_{b'^f}| \tag{8.3d}$$

Therefore, the MILP for partial unloading of the subproblem is defined as Equations 6.7, Equations (6.3a) to (6.3e), Equation (6.3g), Equation (6.3j), Equation (6.3k), Equation (6.13) and Equations 8.3.

This approach means that all batches considered in the subproblem will be completed and that the layout will be configured such that the next batch can be completed directly. However, looking one step ahead is not enough to create good solutions. The final layout will have to be evaluated an optimised on how promising it is for future batches. This will be explained in the next section

8.3.2. Layout quality

In the rolling horizon two things need to be optimised: The amount of crane moves, and the quality of the final layout. In this section, the latter is described. Three principles are introduced to describe the quality of the layout: The components blocked on departure, the amount of unavailable space and the amount of clustering. The components blocked on departure principle will be discussed first.

The idea of blocked components was described earlier. Due to neighbouring components, it might be illegal to retrieve certain components before moving other ones. This is the main cause of the CRP not being trivial. In a given layout of components it is quick to see which components are blocked. In Figure 8.2 a layout is shown where component 1 and 4 are blocked. If these components are to be loaded onto the vessel in ascending order, relocations will be needed to retrieve item one since it is blocked by component 2 and 4 which are to be retrieved later. Therefore, component 1 is said to be blocked on departure. Component 4 is blocked as well, but all surrounding blocks will depart before this. Upon unloading component 4, all surrounding components will already be removed. Component 4 is therefore not blocked on departure.

As shown, a component is to be called blocked on departure if it is blocked directly by an item which will depart at a later time than itself. Indirect blocking occurs when a component further away than direct neighbours cause blocking, like shown in Figure 8.3. Here it can be

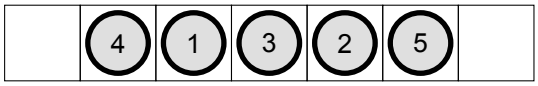


Figure 8.3: Component 1 and 2 are blocked directly while component 3 is blocked indirectly

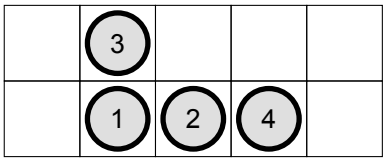


Figure 8.4: The minimum amount of components blocking on arrival is 1, namely component 3.

seen that relocations have to happen before component 3 can be removed, since component 4 and 5 block it. This however, is an indirect block.

In the two dimensional geometry, it is possible that one component blocks multiple other components. Therefore, it is more interesting to consider the blocking components instead of the blocked ones. The minimal amount of blocking components in a layout gives a lower-bound for the reshuffling moves needed for departure. Therefore, the variable f_i is introduced. If a component at position i is blocking another component on departure, it value is equal to 1, else to 0. This is then added for all relocation constraints.

Variable	Description
c	Decision variable: Clustering
f	Decision variable: Blocking
N_b^{d+}	All components departing later than batch b
p^u	penalty for unavailable space

Table 8.2: Variables used in Section 8.3.2

The values are set by defining the minimal values of f in inequalities and by adding f to the cost function. Two things are achieved by this. Firstly, f is always set to a lower bound of moves needed. This can be seen in Figure 8.4. For component 1 it is set that component 3 is blocking. For component 2, either component 3 or component 4 would have to be relocated. Therefore, in this configuration, component 3 and either component 3 or component 4 or blocking. Minimizing the amount of blocking components means that only component 3 is set as blocking. The lower bound for loading all components is than (ignoring loadout restrictions) the amount of components plus the minimum of blocking components, which equals to 5.

The cost function from Equation (6.3a) is modified to account for both the moves done and the blocking components at the end, resulting in Equation (8.5a). Subsequently, the constraints will be added to define f. Before adding these constraints, the component set N_b^{d+} is introduces as shown in Equation (8.4). This set contains all components which are loaded after batch b has been completed.

$$N_b^{d+} = \{n | b \in N_{b'}, b' > b, c_{b'} = 1\} \tag{8.4}$$

First, f is set for front neighbour blocking constraints. This is done in Equation (8.5b). This equation implies that if there is a component at the back row and there is a component which departs later above it, the f value of this component has to be at least one.

Equation (8.5c) is added for all locations with 2 neighbours. Here, for each departing batch b and location i with 2 neighbours, the inequality is added to set the blocking variable f. The first two terms are the occupation-terms for the neighbours, for all components departing later than batch b. The third term is the occupation term for i for all components in batch

b. If all these three terms are 2, it follows that one f of the neighbours has to be at least one. By the same principle, this can be added for the other blocking constraints as well. In Equations (8.5d) and (8.5e), this is added for the locations with 4 relevant side neighbours and Equation (8.5f) contains the constraints for locations with 4 relevant side neighbours.

The concept explained above evaluates the placed components on quality for departure. There is however another important mechanism in deciding the quality: The space available. Consider the layout in Figure 8.5. Regardless of the loadout order, there are no blocked components here. However, this is still a low quality configuration since all unoccupied places are blocked by the two components. These blocked unoccupied spaces are called unavailable spaces, and to improve the rolling horizon a cost is added for these unavailable places.

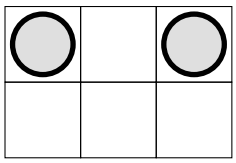


Figure 8.5: Layout with all empty spaces being unavailable

Since only unoccupied locations can be marked as unavailable, the same variable f can be used for this. The equations are similar to those of the constraints for blocking. Since it might be desired to put more priority onto either unavailable locations or blocking components, the cost p^u is introduced, representing the cost of an unavailable location compared to the cost of a blocking component.

For the locations with one location in front of it, the constraints for unavailable locations are shown in Equation (8.5g). The first and second terms are p^u times the occupation binary for the up and down location respectively. This implies that if the location above i is occupied and i is not, f_i has to be at least p^u. For components with two neighbours this is done in Equation (8.5h). Here if the side neighbours of i are occupied and i isn't, then f_i is forced to be at least p^u. The same principle is used for the locations with three and four neighbours, with one equation for each combination of the neighbours left and right. This is shown for three and four neighbours respectively in Equations (8.5i) and (8.5j)

$$\min \sum_{t\in T}\sum_{n\in N}\sum_{i\in S}\sum_{j\in S} x_{tnij} + \sum_{i\in S} f_i \tag{8.5a}$$

$$\sum_{n\in N_b^{d+}} y_{t^e n R_i^h} + \sum_{n\in N_b} y_{t^e n i} - f_{R_i^h} \leq 1 \forall i \in M^h, b \in B^d \tag{8.5b}$$

$$\sum_{n\in N_b^{d+}} y_{t^e n R_{1i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i1}^s} + \sum_{n\in N_b} y_{t^e n i} - f_{R_{1i1}^s} - f_{R_{2i1}^s} \leq 2 \forall i \in M_2^s, n \in B^d \tag{8.5c}$$

$$2\sum_{n\in N_b^{d+}} y_{t^e n R_{1i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i2}^s} + 2\sum_{n\in N_b^{B}} y_{t^e n i} - 2f_{R_{1i1}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_4^s \tag{8.5d}$$

$$2\sum_{n\in N_b^{d+}} y_{t^e n R_{1i2}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i2}^s} + 2\sum_{n\in N_b^{B}} y_{t^e n i} - 2f_{R_{1i2}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_4^s \tag{8.5e}$$

$$2\sum_{n\in N_b^{d+}} y_{t^e n R_{1i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i1}^s} + \sum_{n\in N_b^{d+}} y_{t^e n R_{2i2}^s} + 2\sum_{n\in N_b^{B}} y_{t^e n i} - 2f_{R_{1i1}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_3^s \tag{8.5f}$$

$$y_{t^e R_i^h} - y_{t^e i} - \frac{1}{p^u} f_i \leq 0 \forall i \in M^h \tag{8.5g}$$

Figure 8.6: Configuration showing the need of a clustering coefficient

$$y_{t^e n R^s_{1i1}} + y_{t^e n R^s_{2i1}} - y_{t^e n i} - \frac{1}{p^u} f_i \leq 1 \forall i \in M^s_2, n \in B^d \tag{8.5h}$$

$$2y_{t^e R^s_{1i1}} + y_{t^e R^s_{2i1}} + y_{t^e R^s_{2i2}} - 2y_{t^e n i} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_3 \tag{8.5i}$$

$$2y_{t^e R^s_{1i1}} + y_{t^e R^s_{2i1}} + y_{t^e R^s_{2i2}} - 2y_{t^e n i} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_4 \tag{8.5j}$$

$$2y_{t^e R^s_{1i2}} + y_{t^e R^s_{2i1}} + y_{t^e R^s_{2i2}} - 2y_{t^e n i} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_4 \tag{8.5k}$$

An alternative model can be used where instead of minimizing the blocking components, the amount of blocking components is required to be zero. This forces each problem to finish without any blocked components regardless of the cost. It is done only for the components blocked on departure and not for the unavailable locations. The reasoning behind this is that an unavailable location does not necessarily result in an extra move, while a component blocked on departure will. The model is obtained by replacing Equations (8.5b) to (8.5f) by Equations (8.6a) to (8.6e). In the equations the parts with f in it are removed. This removes the ability to validate the constraints by setting and f and therefore no blocked components are possible.

$$\sum_{n \in N^{d+}_b} y_{t^e n R^h_i} + \sum_{n \in N_b} y_{t^e n i} \leq 1 \forall i \in M^h, b \in B^d \tag{8.6a}$$

$$\sum_{n \in N^{d+}_b} y_{t^e n R^s_{1i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i1}} + \sum_{n \in N_b} y_{t^e n i} - \leq 2 \forall i \in M^s_2, n \in B^d \tag{8.6b}$$

$$2 \sum_{n \in N^{d+}_b} y_{t^e n R^s_{1i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i2}} + 2 \sum_{n \in N^B_b} y_{t^e n i} \leq 4 \forall b \in B^d, i \in M^s_4 \tag{8.6c}$$

$$2 \sum_{n \in N^{d+}_b} y_{t^e n R^s_{1i2}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i2}} + 2 \sum_{n \in N^B_b} y_{t^e n i} \leq 4 \forall b \in B^d, i \in M^s_4 \tag{8.6d}$$

$$2 \sum_{n \in N^{d+}_b} y_{t^e n R^s_{1i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i1}} + \sum_{n \in N^{d+}_b} y_{t^e n R^s_{2i2}} + 2 \sum_{n \in N^B_b} y_{t^e n i} \leq 4 \forall b \in B^d, i \in M^s_3 \tag{8.6e}$$

The final layout quality determiner is called the clustering coefficient. Since the approach of identifying unavailable empty locations only considers the direct neighbours, a layout as shown in Figure 8.6 has no unavailable empty locations according the formulation given in Equations (8.5g) to (8.5j). However, it can be seen that there is one available location in this layout. As soon as a component is placed on either of the empty locations, the other one becomes unavailable. This would not happen if both components in Figure 8.6 were placed next to each other.

For this reason the clustering coefficient c is added. The goal of this coefficient is to decrease the costs of layouts where components are clustered together. The clustering coefficient c_i defines a clustering bonus for every location i. This score, multiplied with the

clustering coefficient p^c, is subtracted from the cost function resulting in Equation (8.7a). The set R_i is defined as the set of all neighbours of location i. This coefficient has an upper bound of zero when there is no component at location i, and an upper bound of $|R_i|$ if there is one. This is set by Equation (8.7b). When there is a component at location i, c_i should be equal to the amount of adjacent components. This is added in Equation (8.7c). Thus by adding Equation (8.7), the optimisation algorithm will have a preference for adding components adjacent to each other. By setting p_c not too large it can be made sure that this does not cause blocking.

$$\min \sum_{t\in T}\sum_{n\in N}\sum_{i\in S}\sum_{j\in S} x_{tnij} + \sum_{i\in S} f_i - p^c \sum_{i\in S^s} c_i \tag{8.7a}$$

$$c_i \leq |R_i| * y_{t^e i} \forall i \in S^s \tag{8.7b}$$

$$c_i - \sum_{j\in R_i} y_{t^e j} \leq 0 \forall i \in S^s \tag{8.7c}$$

Therefore, the MILP the subproblem is defined as Equations 6.7, Equations (6.3b) to (6.3e), Equation (6.3g), Equation (6.3j), Equation (6.3k), Equation (6.13), Equations 8.3, Equation (8.6) and Equation (8.7). This mode limproves the rolling horizon heuristic by measuring the quality of a given layout. By taking into account information about future batches, the algorithm is allowed to look further than it's planning horizon. It turns out however that even the subproblem takes a long time to solve. In the next chapter a method is described on how to increase the solving time by using a warm start.

8.4. Warm start

An MILP solver explores the searchspace by branching on integer variables. Each branching create nodes and for each node, if feasible, an optimal solution is calculated while relaxing the integer requirement for the non-branched variables. If this solution is completely integer, a feasible solution is found for the problem. When this happens all nodes with a (non-integer) optimal solution of a higher value than this solution will be pruned. By pruning a certain node, further exploration for the branch of that node will not happen anymore. Therefore, an integer solution reduces the search space and speeds up the solver.

Instead of letting the solver find the first integer solution, it is also possible to already start with an integer solution. This is called a warm start. The solution does not have to be optimal, but the better it is the more nodes it will prune and the lesser time the solver will take. This warm start is provided for each subproblem. Furthermore, using a known solution will also decrease the search space by reducing the size of the problem, due to the fact that a better upperbound on the amount of moves decreased the amount of variables.

The reason for this is that time is discretised and all timesteps available are stored in T. The amount of both variables and constraints are then directly related to $|T|$. To solve the problem, T has to be large enough to contain the steps needed. An estimate has to be made in the beginning of the size needed for T. If an initial solution is already present, taking T as the time needed for this solution will always give a feasible solution.

If no initial feasible solution is present, T has to be estimated. In practice this will result in making a conservative estimate for T, running the solver, and if no feasible solution can be found the size of T is increased and this process is repeated. This requires the solver to run multiple times for a single problem, which will not happen with a warm start. In the next section it is described how this warm start is generated.

8.4.1. Warm start algorithm

The warm start algorithm divides the subproblem further in non-overlapping problems. Most of these problems are solved by the same LP-model as for the problem, although for the unloading phase a separate algorithm is tried first. This will be described later. The rolling horizon subproblem (RHS) is divided into multiple warm-start subproblems (WSS).

For each batch in the RHS, a WSS is created to complete this batch. Before creating this problem the available place on loadout will be evaluated. If there is not enough space, a WSS will be created without any batches but with the only goal to clear the loadout space. This is possible because of Equation (8.3c). After this problem is solved, a WSS is created and solved to (un)load all components in batch b.

For an arriving batch b, this consists of unloading the components from the vessel to the storage area. These components are placed on either the loadout locations or the storage-only locations. The condition for the final component layout depends on the next batch. If the next batch is an arriving batch, it is required that there will be enough empty locations in the loadout area by Equation (8.8a). The same criterion holds if the next batch is a departing batch, although here the required space in the loadout should be enough just for all components not currently in the loadout area which have to depart in the next batch. This is set by Equation (8.8b).

Variable	Description
b	Batch considered in WSS
$b+1$	Batch directly after WSS
N^b	Components in batch B

Table 8.3: Variables for Equations 8.8

$$\sum_{n \in N} \sum_{i \in S^{sl}} y_{t^e ni} \leq |S^{sl}| - |K^{b+1}| \tag{8.8a}$$

$$\sum_{n \in N \setminus N^b} \sum_{i \in S^{sl}} y_{t^e ni} \leq |S^{sl}| - |K^{b+1}| \tag{8.8b}$$

By combining all moves from all warm start subproblems, it might result in double moves where a component moves to a position in the storage only area $i \in S_s$ and consecutively to another position $j \in S$. Moves like these are detected and contracted. This does not only create a better solution, but also is required because otherwise Equation (7.7) will invalidate the warm start.

Algorithm 9 Warm start algorithm

```
b = 0
moves = {}                                              ▷Batch index
B' batches in subproblem
while b < |B'| do
    if batch type is arriving and Not enough place in loadout then
        new_moves = wss_create_loadout_space(b)
        moves = moves ∪ new_moves
    end if
    new_moves = wss_complete_batch(b)
    moves = moves ∪ new_moves
    b = b + 1
end while
if type of b is arrival and next batch is departure then
    new_moves = wss_prepare_loadout_for_next(b)
    moves = moves ∪ new_moves
end if
moves = contract_double_relocations(moves)
```

8.4.2. Direct unloading algorithm

During the execution of this algorithm, it was shown that the warm start horizon algorithm for arriving batches took a long time. During this time two problems are solved simultaneously:

Where to place the components and how to place them there. If separated, the former can be solved fairly quick if no other components are relocated.

The following approach is therefore tried for arriving components. Consider all arriving components N^a in batch b. A model can then be made to define a location for each component, while optimizing the layout quality. All other components $N \setminus N^a$ are considered fixed. The model to define the component position is based on a relaxed version of the repositioning model, without the blocking constraints and variables. Instead, the binary variable z_{ni} is used. For arriving components, it holds that $z_{ni} = 1$ if component n is placed at location i, and 0 otherwise. For components already in the storage area, z_{ni} is fixed at 1 if component n is initially located at location i. The model will assume direct unloading, without reshuffling.

If this model can be solved, unloading to these positions can be done by a simple algorithm, which places the components in the correct order. This method finds a feasible solution for the unloading part in a fraction of the time needed to solve the relocation model. Afterwards, the relocation model is still ran to prepare the field for the next batch by transferring components between storage only area and loadout area.

As stated before, the location model uses the binary variable z_{ni} , being equal to 1 if a component n is moved to i. Furthermore, it also includes the layout quality variable f. The variable z_{ni} is only defined for the components N_b^B in arriving batch b. The cost function is defined as the cost based on cost variable f, which is given in Equation (8.9a). The equations Equations (8.9b) and (8.9c) impose that at each location only one component can be placed and that all components have to be relocated to the storage-only locations, respectively.

Furthermore, two types of locations are identified. Locations in enclosed clusters and locations in free clusters. A cluster is defined as a set of adjacent, unoccupied and unblocked locations in the same row. If the cluster is enclosed on both sides by a component, it is called an enclosed cluster. Otherwise it is called a free cluster. This is shown in Figure 8.7. Here all non-blocked positions are divided into either free or enclosed clusters.

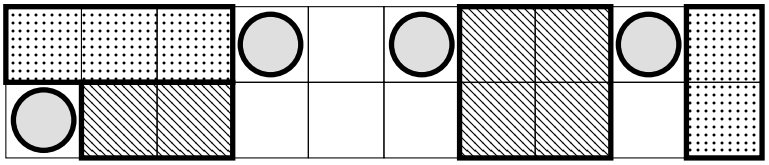


Figure 8.7: Field with components (gray), enclosed clusters(stripes) and free clusters(dots)

The set of all clusters is called C, with C^f denoting the free clusters and C^e denoting the enclosed clusters. S_r^c includes all locations inside cluster r. Without any relocations, a free cluster can be filled completely with components. An enclosed cluster of length $|S_r^c|$ can include at most have $|S_r^c|-1$. This is because of the side neighbour constraints. It can be seen in Figure 8.7 that as soon as one component is added to an enclosed cluster, it is impossible to add a second one without reshuffling. There will be always one unoccupied location left.

Therefore, Equation (8.9d) is added. This equation sets the total amount of components placed in an enclosed segment to maximum $|S_r^c| - 1$. A layout restricted by these constraints can therefore always be placed without reshuffling. Furthermore, the constraints which define the costs of blocking components and unavailable locations are added. This is done by the usual constraints for this, modifying them slightly for the z variable. This is shown in Equations (8.9e) to (8.9p).

In this model all components are removed to the storage-only locations, but the components already placed in the loadout area do not move. It therefore might be possible that there is not enough space for the next batch. If this is the case, the normal WSS will run for relocating these components.

Variable	Description
S'	Locations unoccupied at beginning
N^u	Movable components
R	all clusters
R^f	All free clusters
R^e	All enclosed clusters
S_r^c	Locations of all clusters in S_r^c
b^u	Current batch

Table 8.4: Variables for Equations 8.9

$$\min \sum_{i \in S^u} f_i - p^c \sum_{i \in S^s} c_i \tag{8.9a}$$

subject to

$$\sum_{n \in N^u} z_{ni} \leq 1 \forall i \in S^u \tag{8.9b}$$

$$\sum_{i \in S^u} x_{ni} = 1 \forall n \in N' \tag{8.9c}$$

$$\sum_{n \in N^u} \sum_{i \in S_r^c} z_{ni} \leq |S_r^c| - 1 \forall r \in C^e \tag{8.9d}$$

$$\sum_{n \in N_b^{d+}} z_{nR_i^h} + \sum_{n \in N_b} z_{ni} - f_{R_i^h} \leq 1 \forall i \in M^h, b \in B^d \tag{8.9e}$$

$$\sum_{n \in N_b^{d+}} z_{nR_{1i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i1}^s} + \sum_{n \in N_b} z_{ni} - f_{R_{1i1}^s} - f_{R_{2i1}^s} \leq 2 \forall i \in M_2^s, n \in B^d \tag{8.9f}$$

$$2 \sum_{n \in N_b^{d+}} z_{nR_{1i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i2}^s} + 2 \sum_{n \in N_b^B} z_{ni} - 2 f_{R_{1i1}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_4^s \tag{8.9g}$$

$$2 \sum_{n \in N_b^{d+}} z_{nR_{1i2}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i2}^s} + 2 \sum_{n \in N_b^B} z_{ni} - 2 f_{R_{1i2}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_4^s \tag{8.9h}$$

$$2 \sum_{n \in N_b^{d+}} z_{nR_{1i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i1}^s} + \sum_{n \in N_b^{d+}} z_{nR_{2i2}^s} + 2 \sum_{n \in N_b^B} z_{ni} - 2 f_{R_{1i1}^s} - f_{R_{2i1}^s} - f_{R_{2i2}^s} \leq 4 \forall b \in B^d, i \in M_3^s \tag{8.9i}$$

$$\sum_{n \in N} z_{nR_i^h} - \sum_{n \in N} z_{ni} - \frac{1}{p^u} f_i \leq 0 \forall i \in M^h \tag{8.9j}$$

$$\sum_{n\in N} z_{nR^s_{1i1}} + \sum_{n\in N} z_{nR^s_{2i1}} - \sum_{n\in N} z_{ni} - \frac{1}{p^u} f_i \leq 1 \forall i \in M^s_2, n \in B^d \tag{8.9k}$$

$$2\sum_{n\in N} z_{nR^s_{1i1}} + \sum_{n\in N} z_{nR^s_{2i1}} + \sum_{n\in N} z_{nR^s_{2i2}} - 2\sum_{n\in N} z_{ni} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_3 \tag{8.9l}$$

$$2\sum_{n\in N} z_{nR^s_{1i1}} + \sum_{n\in N} z_{nR^s_{2i1}} + \sum_{n\in N} z_{nR^s_{2i2}} - 2\sum_{n\in N} z_{ni} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_4 \tag{8.9m}$$

$$2\sum_{n\in N} z_{R^s_{1i2}} + \sum_{n\in N} z_{nR^s_{2i1}} + \sum_{n\in N} z_{nR^s_{2i2}} - 2\sum_{n\in N} z_{ni} - \frac{2}{p^u} f_i \leq 2 \forall i \in M^s_4 \tag{8.9n}$$

$$c_i \leq |R_i| * \sum_{n\in N} z_{ni} \forall i \in S^s \tag{8.9o}$$

$$c_i - \sum_{j\in R_i} z_{nj} \leq 0 \forall i \in S^s \tag{8.9p}$$

$$z_{ni} \in \{0,1\} \forall n \in N, i \in S \tag{8.9q}$$

Although this algorithm will quite often not provide a feasible solution, it's computational time is negligible compared to running the full MILP model. The role of this algorithm is thus to see quickly if a simple solution is possible to reduce the amount of times the full MILP model has to run.

8.5. Discussion

Multiple remarks can be made for this solution approach. It will be shown later that although this model performs in a reasonable time compared to project execution duration and makes it possible to find solutions for which the complete MILP model could not find any, large instances will still take a considerable amount of time. A possible way of improving this model would be to consider certain components to be fixed.

For a single horizon step, all components are considered. Although it is certainly possible that it might be beneficial to move components without them being required to move at that time, it also increased the problem size. If a faster method is required, it might be interesting to see if certain components can be considered immovable at certain timesteps. This would decrease the problem size. A possible way of deciding on the immovable components would be by considering the components which are non-blocking and depart later than the components in the current batch. It is probably useful though to still consider blocking conditions on immovable components.

The direct loadout algorithm tries to place all components directly to the storage-only field. This is done because the goal of the warm start is to create a feasible solution and moving all components to storage-only simplifies the future steps. However, there are two issues with this. First of all no solution will exists if there is not enough space in the storage-only area, which results in the full MILP model having to be solved for the arriving batch. This will especially decrease performance in configurations with relatively more loadout places. Secondly, leaving components in the loadout area might result in less moves, especially when these components will be requested to leave the storage area shortly after arriving.

The final discussion point is the upper-bound on the amount of moves on subproblems. Currently, for the warm-start subproblems, this is taken as slightly higher than the lower-

bound, and if no solution is found this upper bound is increased. For the rolling-horizon subproblems, this upper bound is taken as the value from the warm start solution. A tight upper bound does decrease the solver-time, but since optimisation of these subproblems is done for both quality and the amount of moves, having a tight upper bound might also cut away optimal solutions. For example, consider a solution which uses exactly the amount of moves allowed. If there is a possible move which will increase the quality, for example by unblocking a component, doing this move might lead to a better solution. However, due to the upper bound on the amount of moves, this solution does not fit inside the MILP model.

8.6. Conclusion

The rolling horizon algorithm makes it possible to use the earlier given MILP model for real world installation problems. By using the MILP formulation the algorithm can be easily modified similar problems as long as an MILP formulation can be found. This is important since the relocation of TPs is a specific problem, but relocating components in a two dimensional space with spatial constraints is a problem which occurs on multiple occasions during wind farm installation projects or harbour operations in general. Besides the generality introduced by using the MILP formulation, it also allows incorporation of future research on similar problems as well as improvements in MILP solvers in general.

There are some tuning parameters available. The length of the horizon defines the size of a horizon-subproblem, and thus the solver duration. The choice whether to allow blocking components can be seen as the choice of solving a problem directly, or when needed. These parameters will allow the user the find a balance in feasibility, performance and duration. Additionally, due to the structure of the algorithm, it allows the user to save computational time by only solving the problem up to a certain time. By doing this, it still takes into account information about future arrivals and departure in the form of blocking constraints, while not spending computational time on unneeded moves.

In the remaining chapters, a model and method for the SIRP will be given. After this computational results for all solution methods are presented.

9

Ship Installation Routing Model

In the SIRP, the ship routes and task are assigned. There are two principles, which are defining in solving this problem. First, there is the availability consideration. Defining a ship route might put requirements on certain other objects. When two ships have to be on location simultaneously to perform an action, it means that this action can only performed if both ships are available. Secondly, there is the travelling consideration. A ship has to travel between turbines, and it is desired to minimise this travelling time.

Both principles affect each other. A schedule optimised on waiting time might result in an inefficient travelling path, and an optimal route based on sailing distances might result in waiting time due to synchronization. In the analysis of the Walney project, it was shown that it happened often that an installation ship was waiting on a monopile to arrive. By modifying the installation order, this can be prevented, although this might result in a large increase in intrafield travelling time. Therefore, the optimisation must take into account both travelling distances and synchronization.

Two modelling methods have been considered. The first option takes a vehicle routing problem and expands it to include time synchronization. The second option models the problem as a flowshop, and adds travelling to it. Both approaches were evaluated based on available literature, and for the flowshop based approached a large gap in the literature was found, as is discussed in Appendix B. For this reason, the VRP model is used. This model is described in Section 4.3.2. The basic version consists of a network with one depot node and multiple customers node. The goal here is to route multiple vehicles so that each node is visited exactly once for the lowest total travelling cost.

In this chapter, a vehicle routing model is given for the SIRP in multiple steps. In the first step the basic VRP-model is given for the ship routing problem. Subsequently, the principle of multiple installation methods are given with installation precedence. After this the plugs are added and the availability of components.

9.1. Installation routing

In this section, the single-ship installation model is given for the SIRP. In this simplified problem, a turbine can be installed in a single visit by any ship. Set K contains all ships and $K^i \subseteq K$ contains the installation ships. In this problem, it thus holds that $K = K^i$. Each ship k has a capacity Q_k, meaning that it can visit Q_k turbines in a single trip.

The field is defined as a directed graph with nodes N. Any trip starts in node n^0, representing the harbour. Subsequently, a subset of the turbine nodes N^t is visited, before returning to the returning harbour node n^e. This replicate harbour node is added to have a distinction between the arrival and departure time, as will be shown later.

The network is defined as having an arc (n^0, i) for each turbine i, arcs in both directions between any two turbines, an arc (i, n^e) for every turbine to the return harbour node and, finally, there is an arc (n^0, n^e), which enables a certain ship to not travel on a trip.

The cost is defined by a three-dimensional matrix c. For each arc (i,j) and ship k, the cost c_{kij} is defined as the activities done at i, plus the travelling from i to j, for ship k. If $i = n^0$, this includes the time of loading and travelling from the harbour to the turbine. If $i \in N^t$, the installation time is added. The cost of arc (n^0, n^e) is 0, as this describes a ship not travelling. If an arc (i,j) does not exists, the cost c_{kij} is set to a very large number H. These variables were found in multiple articles about vehicle-dependent durations, for example in Stålhane et al. [2015].

A vehicle is allowed to make multiple trips to the installation field. Tests were done with both a three-index formulation and a four-index formulation, as found in Aghezzaf et al. [2006] and Cattaruzza et al. [2016], respectively. In a three-index formulation, x_{kij} is 1 if and only if arc (i,j) is traversed at any point by ship k. In a four-index formulation, x_{krij} is 1 if and only if arc (i,j) is traversed by ship k on trip r. Small performance tests were performed for both formulations, which resulted in the choice of the four-index formulation due to smaller solving times.

For the four-index formulation, the set of trip sets R is introduced. Each ship has a set of trips R_k, from n^0 to n^e. If a ship uses less than $|R_k|$ trips, the direct arc from n^0 to n^e is used to represent these unused trips. Furthermore, the arrival time t is introduced. The problem considered by Cattaruzza et al. [2016] did not include time variables, and for a four-index multi-trip VRP no case with time variables has been found. Additionally, the three-index formulation, as found in Toth and Vigo [2002], did not specify the times at the depots. Since, as will be shown later, ship precedence constraints and component availability constraints require the times at the depot to be specified, this approach cannot be used.

Therefore, the vehicle-dependent single-trip time variable formulation, as used by numerous authors (e.g: Bredström and Rönnqvist [2008]), has been modified to account for multiple trips per vehicle. The variable t_{kri} denotes the time that ship k visits i on trip r. If a ship does not make a visit corresponding to (k,r,i), $t_{kri} = 0$. Furthermore, the helper variable y is introduced, where $y_{kri} = 1$ if and only if ship k visits turbine i on trip r. Similarly, $y_{ki} = 1$ if and only if ship k visits turbine i on any trip. Both variables are an expression of x, as seen in Equations (9.1) and (9.2).

$$y_{kri} = \sum_{j \in N} x_{krji} \tag{9.1}$$

$$y_{ki} = \sum_{j \in N} \sum_{r \in R_k} x_{krji} \tag{9.2}$$

Since the ship installation routing problem is optimised for ship and project cost, additional variables have been introduced. Similar to Yu et al. [2017], a variable e is introduced which represents the total project duration. Although not found in literature, this approach can be expanded by also introducing variables ss_k and es_k, which denote the starting time and ending time of ship k, respectively.

Together with the constants sc_k and pc, for ship and project cost respectively, the cost function of the model can be defined as Equation (9.3a).

With the variables in Table 9.1, the MILP model can be formulated. First the routing constraints are given. In a connected trip from n^0 to n^e, the following holds: For both n^0 and n^e, there is exactly one outgoing and incoming arc, respectively. For all other nodes, it holds that the amount of arcs coming in is equal to the amount of arcs going out. The constraints for trips beginning and ending at the harbour nodes are given in Equations (9.3b) and (9.3c) respectively. The constraints for the turbine nodes are given in Equation (9.3d). Furthermore, every node needs to be visited exactly once. This constraint is added in Equation (9.3e). Additionally, Equation (9.3f) is added to limit the maximum amount of turbines per trip to the capacity of a ship.

To complete the routing requirements, the timing variables and constraints need to be added. The timing variable t_{kri} denotes the time that ship k arrives at node i on trip r. If this visit does not exist, $t_{kri} = 0$. If $x_{krij} = 1$, Equation (9.3g) turns into $t_{krj} - t_{kri} \geq c_{kij}$,

Variable	Description
x_{krij}	Binary variable representing if ship k travels over arc (i,j) in trip r
y_{ki}	Helper variable: one if and only if ship k visits node i
y_{kri}	Helper variable: one if and only if ship k visits node i in trip r
t_{kri}	Variable equal to time of arrival of ship k at node i on trip r, or 0 if this visit does not exist.
K	Set of all ships
K^i	Installation ships
Q_k	Capacity of ship k
c_{kij}	Time of activity at i plus travelling along arc (i,j) for ship k
N	Set of all nodes
N^t	Set of all turbine nodes
n^0	departing harbour node
n^e	arriving harbour node
H	A very large number (representing infinity)
e	Ending time of project
ss	Start time of ship
es	Ending time of ship
sc_k	Cost for ship k per time
pc	Project cost per time

Table 9.1: Variables for the single-ship installation routing problem

meaning that the arrival time at j is at least c_{kij} later than the arrival time at i. Conversely, if $x_{krij} = 0$, the corresponding constraint turns into $t_{kri} - t_{krj} \leq H$, where H represents infinity. Therefore, no restriction is put on any variable in this case. This allows Equation (9.3g) to define a relationship only when a certain arc is used, and have no effect otherwise.

If a certain node is not visited in a trip, the corresponding t variable is set to zero. This is done in Equation (9.3h). This completes the timing part for individual trips. This also mitigates subtours by preventing cycles, as shown in Lemma 9.1.1. Besides the timing constraints for nodes in trips, the timing between trips should also be set. This is set in Equation (9.3i). In Equations (9.3j) to (9.3l), the variables e,es and ss are defined by setting them smaller or larger than any beginning or ending time. This allows the cost to be evaluated on makespan.

The variable ranges are defined in Equations (9.3m) and (9.3n) to finalise the MILP model. This results in the MILP model as given in Equation (9.3).

$$\min pc * e + \sum_{k \in K_i} sc_k es_k - \sum_{k \in K_i} sc_k ss_k \tag{9.3a}$$

subject to

$$\sum_{j \in N} x_{kr0j} = 1 \forall k \in K, r \in R_k \tag{9.3b}$$

$$\sum_{i \in N} x_{krin^e} = 1 \forall k \in K, r \in R_k \tag{9.3c}$$

$$\sum_{r \in R_k} \sum_{j \in N} x_{krij} - \sum_{r \in R_k} \sum_{j \in N} x_{krij} \forall k \in K, i \in N^t \tag{9.3d}$$

$$\sum_{k \in K_i} y_{ki} = 1 \forall i \in N^t \tag{9.3e}$$

$$\sum_{i \in N^t} y_{kri} \leq Q_k \forall i \in N^t \forall k \in K, r \in R_k \tag{9.3f}$$

$$t_{kri} - t_{krj} + (c_{kij} + H)x_{krij} \leq H \forall k \in K, r \in R_k, i \in N, j \in N \tag{9.3g}$$

$$t_{kri} - Hy_{kri} \leq 0 \forall k \in K, r \in R_k, i \in N \tag{9.3h}$$

$$t_{krn^e} - t_{k(r+1)n^0} \leq 0 \forall k \in K, r\ R_k \setminus \{|R_k| - 1\} \tag{9.3i}$$

$$t_{krn_e} - e \leq 0 \forall k \in K, r \in R_k \tag{9.3j}$$

$$t_{krn^e} - es_k \leq 0 \forall k \in K, r \in R_k \tag{9.3k}$$

$$ss_k - t_{kr0} \leq 0 \forall k \in K, r \in R_k \tag{9.3l}$$

$$x_{krij} \in \{0,1\} \forall k \in K, r \in R_k, i \in N, j \in N \tag{9.3m}$$

$$t_{kri} \geq 0 \forall k \in K, r \in R_k, i \in N \tag{9.3n}$$

This MILP formulation is a combination of the four-index formulation of Cattaruzza et al. [2016], the network (with separate start and ending node) from Bredström and Rönnqvist [2008] and the timing constraints based on Azi et al. [2007]. Besides combining this, the t variable has been changed to include both a trip and a vehicle index. Additional constraints were added to account for the vehicle duration variables ss and es, and a constraint was added to set t to zero if not used. It will be shown later why this is important. In the next section, this MILP formulation will be expanded to account for installation methods and precedence relationships.

Lemma 9.1.1. *Consider a single trip by a ship. By introducing variable t'_i for each node $n \in N$ and adding Equation* (9.4) *as constraints, cycles are prevented.*

$$t'_i - t'_j + (c'_{ij} + H)x_{ij} \leq H \forall i \in N, j \in N \tag{9.4}$$

Proof. Let S be a cycle include i'. For each arc (i, j) in this cycle $x_{ij} = 1$ and thus by Equation (9.4):

$$t'_i + c'_{ij} \leq t'_j \tag{9.5}$$

Since $c'_{ij} = 0$ if and only if $i = n^0$ and $j = n^e$, and n^e can never be part of a cycle since it has no outgoing arcs, it follows that $t'_i < t'_j$. Thus for a cycle $S = (i_0, \ldots i_n)$ it would imply that $t'_{i_0} < .. < t'_{i_n} < t'_{i_0} \rightarrow t'_{i_0} < t'_{i_0}$, which is impossible. □

9.2. Installation methods and precedence relations

In the MILP model given above, each turbine node gets visited once by any ship. However, installing a turbine takes multiple ships, and often these ships can only perform their actions if a certain other action is done. For example, installing a transition piece can only be done after the foundation is installed. In this section the model will be expanded to account for these characteristics.

For this purpose, the concept of installation methods M is introduced. An installation method $m \in M$ is a collection of method steps $s \in S_m$, and each method step is a set of ships K_s^m. A turbine is installed if one installation method for this turbine is completed. An installation method is completed if every step in it is completed, by having at least on ship of this method visit the turbine.

This is illustrated in the following example. Consider M, S and K^m as given in Equations (9.6) to (9.11). Equation (9.6) shows that there are two installation methods. The first installation methods exists of a single step, with a single ship, as shown in Equations (9.7) and (9.9). Therefore, to install turbine i by this method, it has to be visited by ship 1. The other installation step consists out of two steps, as seen in Equation (9.8). This means that, for installation by this method, one ship from Equation (9.10) and one ship from Equation (9.11) has to visit i. In logical terms, this can be expresses as Equation (9.12), where b_k is true if ship k visits node i and false otherwise. Note that installation methods only define which ships should visit the node, it says nothing about the order or time when this should happen.

$$M = \{1, 2\} \tag{9.6}$$

$$S_1 = \{1\} \tag{9.7}$$

$$S_2 = \{2, 3\} \tag{9.8}$$

$$K_1^m = \{1\} \tag{9.9}$$

$$K_2^m = \{2, 3\} \tag{9.10}$$

$$K_3^m = \{4\} \tag{9.11}$$

$$M : b_0 \vee ((b_1 \vee b_2) \wedge b_3) \tag{9.12}$$

Each turbine has a set of installation methods by which it can be installed. This set for a turbine i is defined as M_i^t. For each turbine, one of these methods has to be completed. To achieve this, first Equation (9.3e) is removed from the model. Subsequently, the variable p is introduced. p_{mi} is a binary variable, equal to one if and only if turbine i is installed by method m. Equation (9.13a) defines that every turbine has to have a compatible installation method selected. Equation (9.13b) adds the requirement that if a method m is chosen for turbine i, for every step at least one ship of that ship has to visit i.

The next characteristic to be included in the model is installation precedence. An installation precedence relationship requires a certain component has to be installed before another one. Until so far, the installation times were included in c, but to include installation precedence, the installation times per ship are defined separately under c_k^i. Furthermore, in Di, the installation precedence relationships are stored. For each precedence relationship $d \in Di$, K_{d1}^D denotes the set of ships, of which one has to visit a turbine before a ship in the set K_{d2}^D.

This is added in Equation (9.13c), by using the notation from Bredström and Rönnqvist [2008]. The first and third term denote the ship visiting time. The ship visiting time can be taken as the sum of t over all trips, due to the fact that t is set to 0 when a ship doesn't visit a node, by Equation (9.3h). Furthermore, it is required for this approach that for every precedence relationship, the turbine has to be visited by exactly one ship from both sets. This can be done by defining the installation methods correctly. The equation sets the time of arrival of the second ship to be later than the arrival of the first ship plus the installation method.

In the Walney project the Aeolus will first install complete turbines by the jacked installation method. After finishing these it will undergo a week-long transformation to install transition pieces with the floating method. In the ship routing model this will be modelled as two separate ships, with a ship precedence constraint. A ship precedence constraint $d \in Ds$ defines that all trips of ship k_{d1}^D must be finished t_d^D time before ship k_{d2}^D can start.

$$\sum_{m \in M_i^t} p_{mi} = 1 \forall i \in N^t \tag{9.13a}$$

$$p_{mi} - \sum_{k \in K_s^m} y_{ki} \leq 0 \forall m \in M, s \in S_m, i \in N^t \tag{9.13b}$$

$$\sum_{k \in K_{d1}^D} \sum_{r \in R_k} t_{kri} + \sum_{k \in K_{d1}^D} \sum_{r \in R_k} \sum_{j \in N} x_{krij} c_k^i - \sum_{k \in K_{d2}^D} \sum_{r \in R_k} t_{kri} \leq 0 \forall d \in Di, i \in N_t \tag{9.13c}$$

$$t_{k_0 r_0 n^e} - d - t_{k_1 r_1 n^0} = 0 \forall (k_0, k_1, d) \in Ds, r_1 = R_{k,|R_k|} r_2 = R_{k,1} \tag{9.13d}$$

$$p_{mi} \in \{0, 1\} \forall m \in M, i \in N^t \tag{9.13e}$$

The modifications to the ship routing model make it possible to include installation with multiple installation method, with both precedence relationships for installation tasks and

Variable	Description
M	Installation methods
M_i^t	Installation methods for turbine i
S_m	Method steps for method m
K_s^m	Ships in method step s
c_k^i	Installation time for ship k
p_{mi}	Decision binary variable: Is turbine i installed by installation method m
Di	Installation precedence sets
Ds	Ship precedence set
K_{dn}^D	nth set of ships of installation precedence set d
k_{dn}^D	nth ship of ship precedence set d
t_d^D	Offset between ship precedence d

Table 9.2: Variables used in Equations 9.13

ship actions. The obtained model is shown in Equations 9.13 and Equation (9.3) without Equation (9.3e). The introduced parameters and variables are given in Table 9.2. In the next section plugs will be introduced along with the required synchronization requirements.

9.3. Synchronization of plugs and component availability

Finally, plugs and component availability are added to the model. A monopile can be transported to the field with a tugboat. A monopile first has to be prepared by fitting two compatible plugs at both ends to make the MP buoyant. After the installation, the plugs will be picked up by another tugboat and returned to the harbour. Here they will be reused for transporting another MP.

These requirements can be modelled by introducing virtual nodes as in Drexl [2007]. Virtual nodes are then introduced per turbine, representing the state for plugs after being removed at the locations. Tugs can be routed to pick up these plugs, by introducing constraints where plugs and tugs have to travel simultaneously over certain arcs. This, however, greatly increases the size of the model and in tests this was shown to have a very negative effect on solver efficiency.

Therefore, a simplified approach is used. Since tugboats are not considered a bottleneck during installation due to their relatively low cost, it is assumed that there are enough tugs. This assumption allow plugs to move around without considering the availability of plugs. Then one way to model plugs is as ships with a capacity of one, with a modified loading time to model the monopile preparation.

However, for a ship with capacity one, it is redundant to require it to return to the harbour after every node. It is more efficient to model it as a ship with unlimited capacity. The travelling time from i to j is then the time travelling from i to the harbour, plus the loading time, plus the time travelling from the harbour to j. This will represent the same trip without any simplifications, but a strong reduction in problem size.

Plugs can thus be modelled as separate ships with an infinite capacity. These are defined in the set K^p. For plugs to be included in the installation, they have to be added to the installation method and a synchronization relationship has to be added. A synchronization relationship requires that from two sets, one ship from each set visits the turbine from at the same time. The exact synchronization sets are stored in De, and for each $d \in De$ it holds that one ship from K_{d1}^E has to be simultaneous with a ship from K_{d2}^E at a turbine. This is set by Equation (9.14a),

A benefit of this technique is that it also allows for the modification of plugs. Although not considered in the modelling scope of this thesis, a plug consists of a core and a ring around it. It is possible to exchange this ring and therefore changing the diameter of the plug. This exchange takes time, but allows the plug to be used for a different set of transition pieces.

This can easily be modelled by varying the cost matrix. If a plug, compatible with MP set A can be modified to be compatible with MP set B, this can be modelled as a single plug compatible with $A \cup B$ with an additional cost for every arc between A and B.

Additionally, the availability of components is added. For this, two types of available times are introduced. The trip starting times t_{rki} define when the trip in which ship k visits ship i can be visited. This is shown in Equation (9.14b). This cannot be added for the plugs, since these are modelled as one trip. Therefore, ti_{ki} is introduced. This defines the earliest time a certain ship can visit a node i. For plugs it is to be set at the arrival time of the components, minus the travelling time from harbour to node. This constraint is added in Equation (9.14c).

Variable	Description
De	Exact synchronization sets
K_d^E	Contains the two sets of ships for synchronization set d
tr_{ki}	Trip starting time for trip where ship k visits node i
ti_{ki}	Earliest time of ship k visiting node i

Table 9.3: Parameters and variables used in Equations 9.14

$$\sum_{k \in K_{d1}^E} \sum_{r \in R_k} t_{kri} - \sum_{k \in K_{d2}^E} \sum_{r \in R_k} t_{kri} = 0 \forall d \in De, i \in N \tag{9.14a}$$

$$\sum_{i \in N} y_{kri} tr_{ki} - t_{kr0} \leq 0 \forall k \in K, r \in R_k, j \in N_t \tag{9.14b}$$

$$y_{kri} ti_{ki} - t_{kri} \leq 0 \forall k \in K, r \in R_k, j \in N_t, \forall i \in N_t \tag{9.14c}$$

The introduced variables are shown in Table 9.3 and the full model can thus be defined by Equations 9.13 and 9.14 and Equation (9.3) without Equation (9.3e).

9.4. Discussion

A few things can be noted about this model. First, it can be seen that it has a very weak LP-relaxation. This is due to the cost being based on makespan. The makespan depends on Equation (9.3g), which can be rewritten to Equation (9.15). Here it is shown that if x_{krij} is not 1, the right side is equal to something times H. Since H is a very large number representing infinity, this will quickly result in the left side of the equation being unbounded. This results in a weak LP-relaxation.

$$t_{kri} - t_{krj} + c_{kij} x_{krij} \leq H(1 - x_{krij}) \forall k \in K, r \in R_k, i \in N, j \in N \tag{9.15}$$

Additionally, the model is limited by the fact the each node can only be visited once by a ship. This is not specified specifically by a constraint, but a side effect of Equation (9.13b). This behaviour is desired however, since allowing the same vehicle to visit the same node multiple times will break the approach of summing over the times to get the visiting time of ship k at node i. This same principle has to be kept in mind when defining the installation precedence en synchronization constraints. Both types of constraints consists of two vehicle sets K_0 and K_1. Since in Equations (9.13d) and (9.14a), the arrival times are defined by summing t over all ships, the model will break if multiple ships from either K_0 or K_1 will visit the same nodes. This restriction can be enforced by defining the installation methods correctly or splitting up the synchronization and precedence sets.

On another note, it is interesting to see where this model would be applicable as well and what the modifications needed would be. The core of the model is routing vehicles to perform

tasks with time synchronization constraints and multiple ways to perform these tasks. The area of offshore maintenance seems very promising for this. In literature, onshore problems as forest operations and road maintenance were found. By abstracting the ship installation problem, the model introduced above might be valuable for multiple other areas.

One might also notice that modelling the plugs as ships with a capacity one means that given a subset of turbines for a plug to visit, the total time will always be the same. It also means that, due to the ratio between intrafield travelling time plus plug preparation time and intrafield travelling, the time between any two turbines is nearly equal. The reason that this approach was taken is because of it's versatility. It was shown already that this methods allows for plug modification as well, and it also allows for exact synchronization with ships with a capacity of higher than one. However, if the scope would be narrowed down to improve performance in cost of versatility, one might look for a simpler way of modelling these plugs.

Finally, the model introduced above allows for the possibility of an offset between ship precedence: If there is a ship precedence relationship, there has to be a minimum amount of time between the final activities of the first ship and the first activities of the second ship. However, on might notice that in the model this offset is always present. If the first ship is not used, there will still be a time gap between the project start and the start of the activities of the second ship.

9.5. Conclusion

In this chapter, an MILP model for SIRP. This model represents a vehicle routing problem, thus defining the routes for the ships during installation. The possible routes are contained in a network of turbine locations and the harbour locations. A feasible solution represents a set of ship routes where for each turbine a set of compatible ships visit to fulfil the installation requirements.

The installation requirements were modelled as multiple methods of which exactly one has to be fulfilled per turbine. An installation method exists of a set of ships which have to visit a turbine, with corresponding time constraints. The different options for installation methods and timing constraints are a necessary generalization for the VRP to model the ship routing problem. So far, no literature has been found with a similar combination of these constraints.

With these generalizations, a combination of routing and installation is possible. This model is therefore useful for offshore wind turbine installation projects and for projects with similar characteristics. This model is however not capable of solving these problems. It will later be demonstrated that even for very small problem the computational time becomes very large. This was expected though as the VRP is an $\mathcal{NP}$-complete [Kumar and Panneerselvam, 2012] problem. In the remaining chapters, exact optimisation techniques for the SRP will be explored, as well as a heuristic method.

10

Optimization method for ship routing model

Like the CRP, the SIRP was modelled in CPLEX as well. As the model introduced in Chapter 9 will later be shown to have very bad solver performance, multiple cutting planes were added. Additionally, a BC algorithm was developed.

10.1. Total time cutting planes

First, the *total-time* cutting planes are presented. The makespan variables e,se and ss are based on the maximum and minimum value of the timing variables. A lower bound for the ship makespans are the total ship trip durations. Similarly, for the total project makespan, the lowerbound is the total trip duration for the ships. For each ship, the total cost of all trips is set lesser than or equal to the makespan for that ship, and the total makespan in Equations (10.1) and (10.2), respectively.

$$\sum_{r \in R_k} \sum_{(i,j) \in P} c_{ij}^k x_{krij} - se_k + ss_k \leq 0 \forall k \in K \tag{10.1}$$

$$\sum_{r \in R_k} \sum_{(i,j) \in P} c_{kij} x_{krij} - e \leq 0 \forall k \in K \tag{10.2}$$

10.2. Trip Symmetry

For each ship, the trips R_k are defined up front. The amount of trips has to be enough for a single ship to install all possible turbines. The downside of this is that multiple possible trips are empty. For a single ship with a fixed set of trips, the empty trips can be located on different positions in R_k. Because of this, there are multiple solutions which can represent the same trip, with just the empty trips at different positions. Similar to symmetry in the CRP, this is defined as *trip-symmetry*

To remove trip symmetry, Equation (10.3) can be added. This requires every trip except the first one to be empty, unless the trip before it is not empty. This moves all empty trips to the end of R. Additionally, during most construction projects the assumption can be made that a ship will always go to the field with full capacity, except for the last one. In Equation (10.4), any trip is set to full capacity if the next trip contains at least one turbine. This cut improves the solver performance, but adds a new assumption and it is to be evaluated per problem is this assumption is valid.

$$\sum_{i \in N} x_{k(r+1)0i} - \sum_{i \in N} x_{kr0i} \leq 0 \forall k \in K, r \in R_k \setminus \{|R_k|\}, \tag{10.3}$$

$$Q_k x_{k(r+1)0k} - \sum_{i \in N} \sum_{j \in N_t} x_{krij} \leq 0 \forall k \in K, r \in R_k \setminus \{|R_k|\}, k \in N_t \quad (10.4)$$

10.3. Field trips

Due to the long outerfield travelling times an optimal non-integer solution will probably contain subtours to avoid using the costly arcs from and to the field. To prevent this, a minimum bound on trips to the field might be beneficial. This is done in the form of capacity constraints as given in Equation (5.1). Two types of constraints are added by Equations 10.5. The first constraint-set defines that for each ship the amount of turbines visited is less than capacity per trip times the amount of trips. The same is done in the second constraint-set for trips from the field to the harbour.

$$\sum_{i \in N} \sum_{j \in N^t} \sum_{r \in K_r} x_{krij} - Q_k \sum_{j \in N^t} \sum_{r \in K_r} x_{krn^0 j} \leq 0 \forall k \in K \quad (10.5a)$$

$$\sum_{i \in N} \sum_{j \in N^t} \sum_{r \in K_r} x_{krij} - Q_k \sum_{i \in N^t} \sum_{r \in K_r} x_{krin^e} \leq 0 \forall k \in K \quad (10.5b)$$

10.4. Waiting variables

Variables can also be introduced to improve performance of the solver. Equations (10.1) and (10.2) put the total travelling time as a lower bound on the project and ship duration. The gap between the real duration and the travelling time is the waiting time. Therefore, variables are introduced to define this gap. Two types are used, waiting-node variables and waiting-arc variables. Waiting variables are used in multiple VRP studies, usually when the goal it to minimise this waiting time [Christiansen et al., 2013]. Although this is not the goal of the SIRP model, waiting variables are still introduced in order to define cutting planes.

The waiting arc variable wa_{krij} defines the waiting time for ship k on trip r on arc (i,j), just before arriving at node j. Similarly, the waiting node variable wn_{kri} defines the waiting time of ship k on trip r before node i.

With these new variables, multiple cuts can be added to the model. First, the waiting arcs variables wa are discussed. If a ship k traverses arc (i,j) on trip r, it should hold that the time between leaving and arriving is equal to the travelling time, plus the waiting time as seen in Equation (10.6). This equation is split up in a larger than and smaller than equation to only impose it when arc (i,j) is used. This can be seen in Equations (10.7a) and (10.7b). In these equations, the right-part is always larger if x_{krij} is zero, while otherwise both terms with H in it cancel out. Furthermore, Equation (10.7c) is added to set zero if arc (i,j) is not being used.

$$c_{kij} + wa_{krij} = t_{krj} - t_{kri} \quad (10.6)$$

$$t_{kri} - t_{krj} + wa_{krij} + Hx_{krij} \leq H - c_{kij} \forall k \in K, r \in R_k, i \in N, j \in N \quad (10.7a)$$

$$-t_{kri} + t_{krj} - wa_{krij} + Hx_{krij} \leq H + c^{kij} \forall k \in K, r \in R_k, i \in N, j \in N \quad (10.7b)$$

$$wa_{krij} - H * x_{krij} \leq 0 \forall k \in K, r \in R_k, i \in N, j \in N \quad (10.7c)$$

$$wa_{krij} \geq 0 \forall k \in K, r \in K_r, i \in N, j \in N \quad (10.7d)$$

After defining the mechanics of the waiting arc variables, they can be used to formulate multiple cuts. Before doing this, however, the variable can be set to zero in multiple cases. First of all, a vessel will not wait before returning to the harbour, so for all arcs $\{(i, N^e) | i \in N^t\}$,

the variable wa is set to zero. Furthermore, wa can be set to zero for all ships, except ships which form the second part of either a ship, or installation precedence relationship, or are part of a synchronization relationship.

For a ship which has no synchronization relationship and is not included in the second part of a precedence relationship, waiting on all intrafield arcs can be set to zero. Furthermore, if a vessel is also not the second part of a ship precedence relationship, all wa can be set to zero for all arcs from harbour to field as well. Otherwise, if it is the second part of a ship precedence relationship, all arcs to the field not in the first trip can be set to zero.

With this, the cuts in Equations (10.1) and (10.2) can be modified to also account for waiting time. This results in Equation (10.8). The same approach can be used for the node waiting variables wn_{kri}. Here, the waiting time is defined per node instead of arc.

$$\sum_{r \in R_k} \sum_{(i,j) \in A} c_{kij} x_{krij} + \sum_{r \in R_k} \sum_{(i,j) \in A} w_{krij} \leq se_k - ss_k \forall k \in K \tag{10.8a}$$

$$\sum_{r \in R_k} \sum_{(i,j) \in A} w_{krij} + \sum_{r \in R_k} \sum_{(i,j) \in A} c_{kij} x_{krij} - e \leq 0 \forall k \in K \tag{10.8b}$$

The node waiting time wn_{kri} defines the time ship k is waiting on trip r just before arriving at node i. This is defined in Equations (10.9a) and (10.9b). Furthermore, Equation (10.9c) sets wn_{kri} to zero if k does not visit node i on trip r. These can be set to zero for all ships with no precedence or synchronization requirements. If a ship only occupies the first place of a precedence relationship the same holds. Furthermore, it can again be set for ships which only have a ship precedence relationship, that wn_{kri} is zero for all $r > 0$. With this, the lower bounds on both project and ship duration can be set as Equations (10.9d) and (10.9e)

$$t_{kri} + c_{kij} + wn_{krj} \leq t_{krj} + H - Hx_{krij} \forall k \in K, r \in R_k, i \in N, j \in N \tag{10.9a}$$

$$t_{kri} + c_{kij} + wn_{krj} + H - Hx_{krij} \geq t_{krj} t_{krj} \forall k \in K, r \in R_k, i \in N, j \in N \tag{10.9b}$$

$$w_{kri} \leq Hx_{krij} \forall j \in N \tag{10.9c}$$

$$\sum_{r \in R_k} \sum_{(i,j) \in A} c_{kij} x_{krij} + \sum_{r \in R_k} \sum_{i \in N} wn_{kri} \leq se_k - ss_k \tag{10.9d}$$

$$\sum_{r \in R_k} \sum_{(i,j) \in A} c_{kij} x_{krij} + \sum_{r \in R_k} \sum_{i \in N} wn_{kri} \leq e \tag{10.9e}$$

10.5. Flow cuts

In a flow based formulation, the resources which are to be delivered from depot to consumer nodes are modelled as a flow. In Baldacci et al. [2004], this was shown for the VRP. The same approach will be illustrated here to understand the principle of flow formulation, before applying it to the SIRP. The example is simplified and shows how the flow formulation works for a single vehicle. In the flow formulation, the depot is a source for flow. This flow travels over the directed arcs used in the vehicle routes from depot to customer nodes. At each node, the demand q_i is consumed. Let x_{ij} be the binary decision variable which is one if and only if arc (i,j) is used by the vehicle, and f_{ij} the flow on arc (i,j). The flow consumed q_i by node i can be modelled by Equation (10.10). Furthermore, the total flow per arc is limited to the capacity Q if the arc is used, and zero otherwise. This is done by Equation (10.11). Equation (10.12) set that if arc (i,j) is used, the flow is at least enough to meet the demand at j.

$$\sum_{j \in N} f_{ji} - \sum_{j \in N} f_{ij} = q_i \forall i \in N_t \tag{10.10}$$

$$f_{ij} \leq Q x_{ij} \forall (i,j) \in A \tag{10.11}$$

$$f_{ij} \geq x_{ij} q_j \forall (i,j) \in A \tag{10.12}$$

The flow can thus only be satisfied if the route does not exceed the capacity constraint. This also shows a possible use for the flow formulation: adding the requirement constraints. However, in the SIRP the capacity constraints are already fulfilled due to the four index x formulation. Nevertheless, the flow formulation in Yaman [2006] shows to improve the formulation quality, and therefore this is added to the SIRP. Yaman gives multiple flow formulations. The two-index flow variable f_{ij} defines for each arc the total flow for all ships, and the three-index flow variable f_{kij} defines for each arc the flow for each ship. Similarly, this can be split further to f_{krij} to specify a flow variable for each arc, ship, trip combination.

Since the SIRP might include vehicles with infinite capacity, the two-index f formulation results in an infinite bound on all edges and therefore did not improve the performance of the solver. The other two formulations will be presented here. The three-index formulation is called *k-flow* and the four-index formulation is called *kr-flow*. First, *k-flow* will be presented. This formulation was taken from Yaman [2006] and was modified for the four-index x variable. In Equation (10.13a) the demand (1) is consumed for flow f_k, if k visits node i. The capacity is set by Equation (10.13b). Here the flow f_{kij} is set to be larger than 1 if arc (i,j) is traversed by k. Furthermore, Equation (10.13c) limits the flow to the total capacity. Here d_i is the demand at node i. For $i \in N_t$, $i = 1$. For the depot nodes $i = 0$.

$$\sum_{j \in N} f_{kji} - \sum_{j \in N} f_{kij} - y_{ki} = 0 \forall k \in K, i \in N_t \tag{10.13a}$$

$$f_{kij} \geq \sum_{r \in R_k} x_{krij} \forall (i,j) \in A, k \in K \tag{10.13b}$$

$$f_{kij} \leq \sum_{r \in R_k} (Q_k - d_i) x_{krij} \forall i \in N, j \in N, \tag{10.13c}$$

Yaman [2006] did not consider multiple trips, and therefore did not use a four-index flow-formulation. This formulation is introduced here by defining the flow variables f_{krij} for each combination of ship, trip and arc. The approach is similar to Equations (10.13a) and (10.13c). Equation (10.14a) sets the flow consumed equal to one if node i is visited by vehicle k on trip r. Also analog to *k-flow* formulation, Equation (10.14b) ensures that the capacity constraint is met and Equation (10.14c) requires the flow f_{ij}^{kr} to be at least 1 if arc (i,j) is used by vehicle k on trip r.

$$\sum_{j \in N} f_{krji} - \sum_{j \in N} f_{krij} = y_{kri} \forall k \in K, r \in R_k, i \in N_t \tag{10.14a}$$

$$f_{krij} \leq (Q_k - d_i) x_{krij} \forall (i,j) \in A, k \in K, r \in K_r \tag{10.14b}$$

$$f_{krij} \geq x_{krij} \forall (i,j) \in A, k \in K, r \in K_r \tag{10.14c}$$

10.6. Branch and Cut

The subtour elimination constraints are one of the mayor difficulties in the VRP. It was shown earlier that subtour constraints need to be defined for each subset of nodes, resulting in an exponential amount of constraints. There are other ways however to eliminate these subtours without including a constraint for every subset. In the original formulation given in Chapter 9, subtours are eliminated due to the constraints on time variables. This principle to use additional variables to eliminate subtours was introduced by Miller et al. [1960].

Unfortunately using these extra variables to prevent subtours give a convenient but weak formulation [Pataki, 2000]. A method often used in modern VRP solvers is column generation where variables are generated during the solver algorithm, however it has been shown in the literature study that no working column generation method has been found yet for the VRP with time synchronization between vehicles. For this reason, the possibilities of row generation are explored. In this method, new constraints are generated during the solver process.

The Branch and Cut algorithm was given in Section 4.5.1. This algorithm is a variation on the standard Branch and Bound algorithm commonly used to solve MILP problems. The Branch and Cut algorithm used in this thesis is similar, except that instead of adding constraints after finding a feasible solution, it adds constraint at each Branch and Bound node. After an non-integer optimal solution x' has been found, new constraints are added to cut away x'. This new constraint thus separates x' from the solution space. The set of capacity constraints (Equation (4.6d)) are defined for each subset of N and thus increase factorial. Therefore, it will quickly become impossible to add them all, and instead a Branch and Cut algorithm is needed to decide which subset of cuts to add.

In the implemented BC algorithm a non-integer optimal solution x' is considered per node. It was shown earlier that the timing constraints prevented subtours in the integer solution. However, in the non-integer solution subtours still can occur. A subtour results in a subset of nodes N' for which the demand is not delivered from the depot. For the standard VRP this can be seen as finding a subset where the total capacity of vehicles entering is insufficient for serving all nodes. However, in the ship installation routing, the demand for a subset is not a constant but depends on the installation methods chosen.

It is therefore first explained how the separation problem works for a traditional VRP. After this, the modifications for the time-synchronization problem are given. An approach for capacity constraints in a Branch and Cut algorithm is given in Blasum and Hochstättler [2002], who introduced multiple cuts. Multiple were tried where the capacity constraints gave the best results. This is used in the BC algorithm.

Blasum and Hochstättler [2002] show how to solve the separation constraint for the capacity based VRP constraints by solving maximum flow problem. A maximum flow problem considers a network with a source and a sink node, and capacities on each arc. The max-flow min-cut theorem [Dantzig and Fulkerson, 1955] states that the maximum flow through this network, bounded by the capacities, is equal to the minimum value cut. A cut is a subset of arcs A, such that two distinct subsets of nodes N_1 and N_2 are created. Each arc in A has one endpoint in N_1 and one in N_2, and there is no arc $a \notin A$ connecting N_1 and N_2

Consider node subset $N' \subseteq N^t$. Assume vehicles with capacities Q_k. Let $\delta^-(N')$ consist of all arcs entering the subset of N'. It follows that all nodes have to be serviced by vehicles coming in on these arcs. The total inflow of components in N' on an arc (i,j) is $\sum_{k\in K}\sum_{r\in R_k} x_{ij}^{'kr} * Q_k$. Note that x' does not have to be integer. The total supply of components delivered to N' is therefore given in Equation (10.15). The separation problem is finding N' which has a supply less than the demand.

$$supply(N') = \sum_{k\in K}\sum_{r\in R_k}\sum_{(i,j)\in\delta^-(N')} x'_{ij} * Q_k \tag{10.15}$$

This can be modelled as a maximum flow problem. Create graph G with the harbour node n^0 and turbine nodes N^t and a sink node n^s. Arcs are defined between any 2 nodes in $\{n^0\}\cup N^t$ with the value of $\sum_{k\in K}\sum_{r\in R_k} x'_{ij} * Q_k$, still considering a demand of 1 per node, regardless of which vehicle delivers. Furthermore, add arc (i, N^s) with a capacity of 1 for each turbine node i.

If there is a flow from n^0 to n^s in G of size $|N^t|$, it follows that the demand is fulfilled for every node due to the arc capacity on (i, n^0). If the maximum flow is less, it holds that the arcs in G' do not have enough capacity to send a flow of 1 to every turbine node. The minimum cut then holds the arcs for which the flow is at full capacity. The nodes on the sink-side of

the $n^0 - n^s$ cut therefore do not have enough supply. This set of nodes is taken as N', and the Equation (10.16) can be introduced to require enough supply for this subset of nodes.

$$\sum_{k \in K} \sum_{r \in R_k} \sum_{(i,j) \in \delta^-(N')} x'_{ij} * Q_k \geq |N^t| \tag{10.16}$$

This method provides a bound, although it does not take into account multiple installation methods nor multiple ships per installation method. Equation (10.16) will only require a supply of 1 per turbine. It also does not take into account any information about which method will be used. Therefore, two approaches have been introduced which can be used with the multiple installation methods.

The first approach, called *branch-cut 1*, uses h'. This variable holds the chosen installation methods for the non-integer optimal solution. The min-flow max-cut problem is then solved for each installation step s_m, for each installation method m and for each ship set $K_s \in s_m$. For installation method m, the demand for each turbine node $i \in N_t$ is set to h_i^m, thus setting the capacity on each arc from a turbine to the sink node as given in Equation (10.17). The maximum capacity per edge to a turbine node is defined by Equation (10.18). The mincut-maxflow problem is then solved by using the Edmonds-Karp algorithm [Edmonds, 1965]. If the max flow is lower than $\sum_{i \in N_t} h_i^m$, N' is defined as the set of turbines on the sink-side of the minimum cut. This set thus has not enough inflow of ships to fulfil the demand for the current installation method m. In this case, Equation (10.19) is added for N'.

$$cap_{iN^s} = h_i^m \forall i \in N_t \tag{10.17}$$

$$cap_{ij} = \sum_{k \in K_s} \sum_{r \; in R_k} x_{ij}^{\prime kr} Q_k \forall i \in N, j \in N, i \neq N^e, j \neq N^e \tag{10.18}$$

$$\sum_{i \in S} \sum_{j \in N \setminus N'} \sum_{k \in K_s} \sum_{r \in R_k} (x_{krij} + x_{krji}) \geq 2 \sum_{i \in N'} h_i^m \tag{10.19}$$

The second branch-cut considers the set K^a. In Equation (10.20), M' is defined as the set of all combinations of steps per methods. Equation (10.21) then defines S^a as a collection of ship sets. Each element of S^a holds the ships from a combination of steps S', where S' contains exactly one method step per method.

This means that for every collection of ships $K' \in S^a$, at least one set of ships is contained for the chosen turbine method. Thus for every $K' \in S^a$, each turbine has to be visited by at least one ship $k \in K'$. Therefore, a minflow-maxcut problem is solved for each $K' \in K$.

Since for each node i, one ship in K' has to visit i, the demand per node is one. This means that the arc capacity between each turbine node and the sink node is equal to 1 (Equation (10.22)). The capacity on each arc to a turbine node is equal to the combined capacity of all ships in K', as shown in Equation (10.23).

$$M' = \prod_{m \in M} S_m \tag{10.20}$$

$$S^a = \{\cup_{s \in S'} K_s^m : S' \in M'\} \tag{10.21}$$

$$cap_{in^s} = 1 \forall i \in N_t \tag{10.22}$$

$$cap_{ij} = \sum_{k \in K'} \sum_{r \; in R_k} x_{ij}^{\prime kr} Q_k \forall (i,j) \in A, i \neq N^e, j \neq N^e \tag{10.23}$$

Similar to branch-cut method 1, a minimum cut smaller than $|N_t|$ indicates a subset $N' \subset N$ for which the demand is not met. Equation (10.24) is then added to fulfil this demand.

$$\sum_{i \in S} \sum_{j \in N \setminus N'} \sum_{k \in K'} \sum_{r \in R_k} (x_{krij} + x_{krji}) \geq 2|N'| \tag{10.24}$$

10.7. Discussion

The Branch and Cut method above are created by taking existing capacity based cuts for the vehicle routing problem and modifying it to the concept of different installation methods. Although it will show later that this method provides good results, an important property of these cuts are lost in it. This is the rounding property. This is explained by the following example. If a subset of nodes with demand a has to be supplied by a vehicle with capacity b, the minimum amount of trips to these nodes is a/b. However, since partial trips are not possible, this lower bound can be rounded up to set the minimum amount of trips to $\lceil a/b \rceil$. This thus provides a tighter bound, but relies on the fact that the capacity is a constant. Since in the SIRP the capacity depends on a variable, rounding is not possible.

11

Adaptive Iterative Simulated Annealing algorithm

Since the VRP is well known to be an $\mathcal{NP}$-hard problem, the exact optimisation cannot be used for large instances. The value of formulating an exact method lays in its ability to evaluate other heuristic solution methods. In this chapter, an Adaptive Iterative Simulated Annealing (AISA) for the SIRP will be presented.

11.1. Algorithm

In metallurgy, the process of annealing involves heating and cooling of the material, to increase the size of its crystals. This process served as an inspiration to create the simulated annealing meta-heuristic, which is used to solve a broad range of optimisation problems. The simulated annealing is a LS based. The basic local search algorithm has the problem that when no improving solution can be found in it's neighbourhood, it will stop. This means that it is not capable of escaping local optima.

The approach of simulated annealing to overcome this problem, is to introduce a temperature variable during its execution. This temperature gradually cools during the local-search iterations. When evaluating a neighbouring solution x', the algorithm will always move from the current solution x to x' if the cost $f(x')$ of this new solution is lower than the cost $f(x)$ of x. If $f(x')$ is larger than $f(x)$, it depends probabilistically on the current temperature T. If T is high, the chance of accepting a worsening solution is high as well, and vice versa. This gives the simulated annealing algorithm the capability of escaping local optima, although it cannot be guaranteed that a global optimum will be reached.

It was shown earlier that a mayor problem in using LS based optimisation were the synchronization constraints. If a reasonable good current solution is considered, a neighbourhood solution might be created by changing one or a few nodes. Consider the current solution x and the neighbourhood solution x', created by performing move m on x. It might very well be that the routes in x' are better than those in x. However, only a few nodes in the solution are changed. The rest of the solution is still optimised for the synchronization constraints in x. Therefore, it is likely that the cost of x' is higher than that of x, due to the synchronization requirements. This results in m being marked unfairly as a bad move.

To give x' a fair chance, a new optimisation process starts after doing move m. The goal of this second optimisation process is to fix the synchronization requirements. This involves a second simulation annealing algorithm which handles the synchronization vehicles. Although this mitigates the problem of a high quality solution x' being dismissed only because it is incompatible with the synchronization requirements in x, it increases the time to evaluate a move m by a large amount.

The amount of moves to be evaluated in the upper level simulated annealing algorithm, from now on denoted by SA', is desired to have as few move-evaluations as possible. For

the bottom level simulated annealing algorithm, from now on denoted by SA'', more move evaluations per iterations are allowed. This means that SA'' will, after selecting a move, evaluate multiple variations of this move and select the best one. In contrast, SA' will only evaluate one variation before deciding whether or not to move in this direction.

The structure of the algorithm is shown in Algorithm 10. It is initialised by setting the upper algorithm temperature to an initial condition and by creating a feasible initial solution. The local search is then ran until some stopping criterion is met. A move is selected, and this move creates a new temporary solution x'_u. This solution might be infeasible. Often, a ship visiting a node is changed to a different ship. If this new ship is part of a different installation method, the new solution is not feasible. Therefore, a repair move is done after this. Subsequently, the repaired temporary solution x'_u is used as the initial solution for the bottom algorithm SA''.

The structure of the bottom local search is similar to the upper local search. The difference is found in the initial solution, the moves done, and the stopping criteria. Just like SA' a move is selected and a temporary solution x'_b is created. This solution is then compared against the current bottom solution x_b. Based on the quality of both solutions and the temperature it is then decided if this move is accepted as the new current solution. Furthermore, if the solution is better than the best solution found so far in SA'' it is stored as the best solution. After each iteration in SA'' the bottom temperature T_b is updated.

When some stopping criterion is met, SA'' stops and returns the best solution found x^*_b. This solution is then compared with the current solution x_u from SA'. Depending on quality and temperature, it can be accepted as the new current solution. Again, if it is the best solution found so far it is stored as well. After this, the temperature is updated and the iteration runs again. This is repeated until some stopping criterion terminates the algorithm, after which x^*_u holds the best solution found.

This gives a description of how the iterative simulated annealing algorithm works. There are still multiple details to be presented, such as the stopping criteria, the moves, creation of the initial solution and more. First, the motivation of this algorithm over other possible candidates will be given. Subsequently, the details for different parts of the algorithm is presented.

11.2. Choice of algorithm

With the structure of the algorithm explained, the motivation for this algorithm can now be presented. As seen in the literature review, there are two main approaches in heuristics for synchronised routing, with multiple variations on these. The first one is a large neighbourhood search which has two actions, a destroy and a repair action. In the destroy action, a large part of the solution is to be destroyed which is then repaired by a heuristic in the repair action. Traditional local search algorithms encounter the problem that most neighbourhoods are infeasible due to the synchronization constraints. The idea behind LNS is to destroy a large enough part of the solution, such that these constraints will be loose and new neighbourhoods will be feasible.

The other main approach was a two-level local search. Here, the problem was split into two parts where traditional neighbourhood moves are used at the top level problem. This, of course, results in conflicts with schedules for other vehicles, so after each move belonging to a top-level neighbourhood, a second optimisation is performed to resolve these conflicts.

To make a choice, it is important to first understand the characteristics of our problem. A major difference with problems found in literature, is that we consider a synchronization problem with multiple trips per vehicle. Although each trip can be modelled as a separate vehicle, this adds the constraint that a vehicle can only start after its predecessor has finished its trip. This creates much more synchronization constraints than found in most problems.

This might cause problems with an LNS algorithm. By destroying a part of the solution and rebuilding it, there is no guarantee that a different solution will be generated. It is possible that the rebuild solution is exactly the same as the original solution, but when the amount of possible solution to which the destroyed solution can be rebuilt is high, this chance is very

Algorithm 10 Iterated simulated annealing algorithm

```
T_u = T_u^0
x_u = generate_initial_solution()
x_u^* = x_u'
while Upper stopping criteria not met do
    m_u = select_upper_move()
    x_u' = do_move(m_u,x_u)
    x_u' = repair(x_u')
    T_b = T_b^0
    x_b = x_u'
    x_b^* = x_b'
    while bottom stopping criteria not met do
        T_b = T_b^0
        m_b = select_bottom_move()
        x_b' = do_move(m_b,x_b)
        if accept(x_b', x_b, T_b) then
            x_b = x_b'
        end if
        if f(x_b') < f(x_b^*) then
            x_b^* = x_b'
        end if
        T_b = update_bottom_T()
    end while
    if accept(x_b^*, x_u, T_u) then
        x_u = x_b^*
    end if
    if f(x_b^*) < f(x_u^*) then
        x_u^* = x_b^*
    end if
    T_u = update_upper_T()
end while
```

small. However, if the problem is very constrained, the amount of possible solutions after rebuilding is lower, and the chance of building a similar solution increases. In a two-level local search, a change is forced, and therefore this algorithm will always generate a different solution after a move.

Our problem also has a large dominant-subjective character. Due to the high costs of the installation ships in comparison to other vehicles it is very clear that these need to be optimised. It would basically never be a smart choice to delay these ships to minimise support-vehicle time. The contrary statement is also true, if a support vehicle can be delayed to minimise installation ship time, very often it will be profitable to do so.

The two-parts character matches very well with a two level search. When a schedule of the installation ships is known, the plug-schedules are very constrained. Simple tests also have been done for both types of algorithms, and the results were more promising for the two level local search. However, it has to be stated that the choice relies more on the logical arguments presented above than on quantitative tests results. The reason for this, is that to create a fair comparison a simple test is not sufficient since there are many neighbourhood structures to define in both. The goal of creating both algorithms was to get a better understanding of the these algorithms in order to derive logical arguments.

In the rest of this chapter, the details of the adaptive iterated simulated annealing algorithm for the vehicle routing problem are presented. The next section will cover the first thing to be done when the algorithm starts: Creating an initial solution.

11.3. GRASP Algorithm

As shown in Algorithm 10, the AISA algorithm explores the search space by moving between feasible solutions. Therefore, an initial feasible solution has to be generated. This solution will then be improved by the local search.

Additionally, it is favourable for an initial solution to have some degree of randomness. The reason for this can be found in the mechanics of local search. A local search method starts at a certain initial solution and keeps moving from neighbour to neighbour. The simulated annealing helps the local search in 'climbing out' of local optima, but only to a certain extent. Therefore, the starting location in the searchspace is a large influencer for the final position.

By using multiple initial solutions, the AISA algorithm can run multiple times, exploring different parts of the search space. For this reason, GRASP, as introduced in Section 4.5.2, is used. This algorithm has a randomness built in, and therefore will create a different solution each time it runs.

GRASP stands for greedy randomised adaptive search procedure, and was used by Salazar-Aguilar et al. [2013] and Ait Haddadene et al. [2016] for synchronised routing problems. It starts with an empty solution, and for each iteration it adds one component, randomly selected from the list of n best options. This is what greedy and randomised stands for. Greedy means taking the best option, not taking into account future choices. It is randomised because it selects randomly from a list of top candidates. For the next iteration, the costs of inserting a node somewhere are recalculated, with the choice made in the previous iteration taken into account. This is what adaptive stands for.

Algorithm 11 Grasp implementation for Installation and Routing Problem

```
function create_initial_solution
    node_list = {(i, ⋃_{m∈M_i} Ks_0^m) : i ∈ N_t}
    R = create_empty_routes()
    while |node_list| > 0 do
        scores = {}
        for all (i,ships) ∈ node_list do
            for all k ∈ ships do
                cost_at_every_position = calc_cost_at_every_position(i,k,R_k)
                (position,cost) = select_best(cost_at_every_position)
                scores = scores ∪ {(i,k,position,cost)}
            end for
        end for
        (i,k,position) = select_randomly_from_top_n(scores)
        R = insert(R,i,k,position)
        node_list = remove_node(node_list)
        if k is first ship in method then
            node_list = node_list ∪ create_new_nodes()
        end if
    end while
    return R
end function
```

The GRASP implementation for the SIRP is given in Algorithm 11. When the algorithm starts, a node list is initialised. This node list holds an entry for each turbine node. Each entry exists of the node and the set of ships included in the first step of each installation method compatible for this turbine. Additionally, a set of empty trips is generated. This holds multiple empty trips per ship, depending on the capacity of that ship and the maximum demand.

Then for each node i and each ship k in the corresponding ship list, the cost for inserting node i at every position in trip-set R is calculated. All non feasible positions, for example due to capacity constraints, are removed and the cost at every position is returned. This thus

contains all positions where node i can be inserted in R_k, and its corresponding costs. From these the best position is selected and stored in the scores list.

When all possible ships are evaluated for every node, the result is a node list which holds for each node a ship to include at a position. This results in the smallest possible cost increase for all options for that node. This list is sorted and one entry is selected randomly from the top n entries. This entry holds a node, a ship and a position where to insert this visit. The trips R are updated by changing the corresponding trip of ship k to visit node i at the specified position.

After this, the selected entry is removed from the node list. If it is the case that this entry defines the first visit to turbine i, the method for i will now be fixed. The node list is then updated by adding an entry for each method step. Each entry consists of the node i and the possible ships in the method step. If there is already a visit to node i, it thus follows that all steps were added to the node list earlier and nothing is added. This process is iterated until the node list is empty, resulting in a feasible solution. .

This completes the GRASP algorithm for the SIRP. It is modified from examples found in literature to take into account the multiple installation methods and trips. With this algorithm an initial route can be found, and by changing the size of the restricted list the randomness can be varied.

The grasp algorithm is used on two occasions. Firstly, in creating the initial solution. Secondly, it is used in the repair move. Here the conflicting ships caused by doing a move in SA' are removed and the node list is repopulated with these ships for the relevant turbines. It is then ran with a list size of 1. This repairs the installation method conflicts, after which the repaired solution will be optimised further.

11.4. Moves

In local search, the neighbourhood of a solution x is defined as the set of all other solution in which x can be transformed by a single permutation. These permutations are called moves, and in order to explore the search space multiple moves were defined. A move thus takes a solution x and transforms it into another solution x'. It is important that the moves connect the complete search space, meaning that for any two feasible solutions x^1 and x_2, there is a sequence of moves to transform x_1 to x^2 and vice versa.

A characteristic of a move is how much it changes a certain solution. Moves which perform small changes have the benefit of not moving too far away from the solution x, which might already be a good solution. Changing x to much will often negate the positive effect of earlier improvement steps. A downside of a small alteration move is that it might get trapped in a local optimum. It might be necessary in these cases to have a move which alters the solution a bit more.

The effectiveness of a move varies with the status of the current solution. It is for this reason that instead of one, multiple moves are used. The different moves will be described in this section. The selection mechanism will be discussed afterwards. Each move takes a set of trips, and relocates one or more nodes in this. A move is restricted to a subset of ships for which it can change the routes based on whether the move belongs to SA' or SA''. A division is made between leading ships and following ships. The leading ships are changed in SA' and the following ships are considered in SA''.

The first move considered is the swap move. This move first selects two compatible ships randomly. Compatible means that the nodes in routes between these ships can be swapped. For SA'' this means that the ships are both included in the same method step. In SA' this does not have to be the case, since the repair step in Algorithm 10 will fix the infeasible solution created by swapping two ships which are not part of the same method step. When two compatible ships are selected, a trip and turbine for the first position is selected. This is done by selecting the node which has the maximum waiting time. An additional swap move is created by changing this step to selecting a random node. A list is then created which contains all nodes on all trips for the second ship.

From this list one entry is selected. For SA'' this is done by evaluating the effect of all, and then selecting the best one. For SA', this would result in running SA'' for each evaluation, so instead it is selected randomly. Now two nodes are selected on two trips: (k_0, r_0, i_0) and (k_1, r_1, i_1). Node i_0 and i_1 are then exchanged between their respective trips, as shown in Figure 11.1.

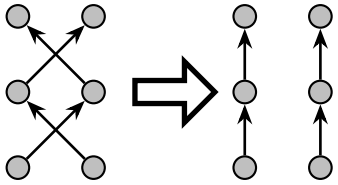


Figure 11.1: Swap move

The second move is called an insert move. The selection mechanic for selecting nodes is the same as for the swap-move, except that for the second ship the final harbour node n^e is considered. This, again, results in two nodes with corresponding ships and trips: (k_1, r_1, i_1) and (k_2, r_2, i_2). Node i_1 Is then removed from r_1 and inserted in r_2 just before i_2. This is shown in Figure 11.2. It is noted that for selection of the second ship and trip, empty trips are considered as well. An empty trip consists of just the beginning and ending harbour node. The only possible position of inserting is then just before the ending harbour node n^e, thus creating a new trip to the field.

The third move considered is the multiswap, which swaps multiple adjacent nodes. It first selects two positions from two compatible trips, similar to the swap and insert moves. The difference is that it does not consider harbour nodes and that it only considers trips with at least two turbine nodes. From both selected positions, a sequence of n nodes is selected, where n is random but will not be larger than the amount of turbine nodes left from the selected position in both positions. These adjacent segments are then exchanged between trips. This exchange is shown in Figure 11.3

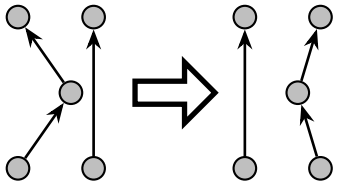


Figure 11.2: insert move

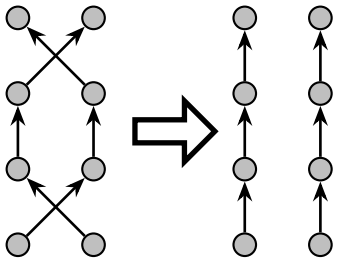


Figure 11.3: Multiswap move

In contrast to the previous moves, the two-opt move does only modify a single trip. The move randomly selects a trip and corresponding ship (k, r). From this trip, one position is selected, either randomly or on longest waiting time. Then a candidate list is created by including all remaining nodes in the selected trip r. Again, if the move is part of SA'' all candidates are evaluated, otherwise one is picked randomly.

Thus two positions i and j in this trip are selected, where i is visited before j. Let i' be the node visited just before i and j' the node visited just after j. The two-opt move then replaces arc (i', i) by (i', j) and arc (j, j') by (i, j'). This can be seen as crossing the arcs to i and from j. Additionally, the arcs from i to j are reversed to maintain a correct route. This move is shown in Figure 11.4.

Furthermore, the cross move is used. This move selects a single ship randomly as long as the ship has at least two trips which both visit more than 1 turbine. Subsequently, two trips are selected and all nodes of the shortest trip are added to a candidate list. The selection of a candidate, which is the same as explained in the previous moves, results in a ship k, two trips for this ship r_0 and r_1, and one node i located on the shortest of these two trips. If i is the nth node in it's trip, then j is defined as the nth node in the other trip. The cross move then crosses the arcs leaving i and j similar to the two-opt move. This is illustrated in Figure 11.7.

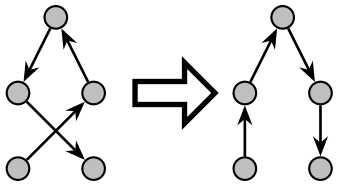


Figure 11.4: Two-opt move

The next move included in the algorithm is called or-opt. This move selects two compatible

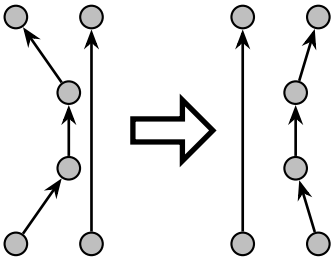


Figure 11.5: Adjacent insert move

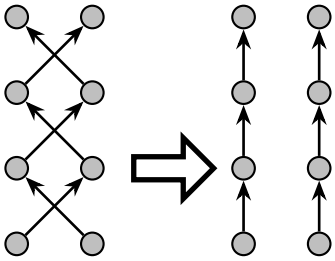


Figure 11.6: Or-opt move

ships with two trips randomly, with both trips visiting at least 3 nodes. The length of the shortest trip is then defined as lr. A random integer from 1 to $\lceil lr/2 \rceil$ is then defined as the amount of nodes n which will be swapped. Then n non-adjacent nodes are selected randomly and swapped with the corresponding nodes in the other trip as shown in Figure 11.6.

Similarly, the adjacent-insert move also selects two trips from two compatible ships. The requirements for the trips are that the first one is not empty ,and that the second one has at least one available space. From the first trip an adjacent segment of random size is selected, with a maximum length equal to the capacity left in the second trip. This adjacent segment is then removed from the first trip and placed at a random position in the second trip. The adjacent insert move can be seen in (11.5).

Finally two more moves are introduced which threat trips as a whole. The reverse move, as seen in Figure 11.8 selects a random trips from any ship and reverses the direction. The trip-swap selects two trip from a single ship and interchanges them. Both these moves will not result in a shorter travelling time, but might improve synchronization and therefore reduce waiting time.

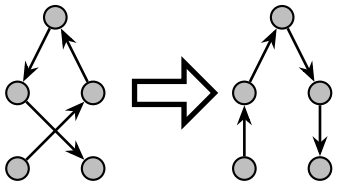


Figure 11.7: Cross move

11.5. Move selection

As stated before, the performance of moves depend on the current solution state. Different moves will be effective at different times during the solving process. For example, the two-opt move has an untangling effect, which is highly effective if a trip has an inefficient ordering of the included nodes. Likewise, when a certain ship has a much longer makespan then another, then the insert move will perform well.

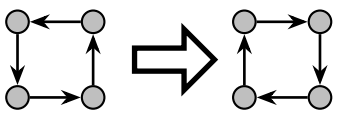


Figure 11.8: Reverse move

Using all moves with the same frequency will therefore often result in selecting moves which are not effective at the current time, therefore wasting computational resources. For this reason, an adaptive move selection mechanism is implemented, as found in Ropke [2005]. This mechanism keeps track of the performance of the moves and uses this information to chance the frequency of choice per move. For each move, two scores are kept, the smoothed score s_m and the observed score s'_m. These scores are updated as the algorithm runs, and choices are made based on the smoothed scores.

All scores are set equal when the algorithm starts. A move is chosen based on the roulette wheel random drawing. This means that the probability $p(m)$ of selecting a certain move m is directly proportional to the score, as is shown in Equation (11.1). For the first iteration, this of course results in an equal probability for all moves. After a move is executed, feedback

is given. Two update parameters are set for the algorithm, the su^a and su^b. If a move is accepted, the observed score s_m for move m will be incremented by su^a. If the move also resulted in a new best solution, it will be incremented by su^b. Furthermore, the amount of move selections for move m are stored in mc_m.

$$p(m) = \frac{s_m}{\sum_{m \in M} s_m} \tag{11.1}$$

This is done for a fixed amount of iterations: The segment length. This is a parameter in the AISA algorithm. At the end of a segment, the smoothed scores are updated according to Equation (11.2). In this equation, two constants are introduced which are both algorithm parameters: The smoothing coefficient ρ and the score constant cs. It can be seen that the new smoothed score is a combination of the previous smoothed score and of the observed score divided by the move call count in the previous segment. The smoothing coefficient defines the weight-factor of both terms. A high ρ will thus result in an observed score resistant against modifications, and a low ρ will result in a smoothed score heavily reliant on the last observed score. Furthermore, introduced in this thesis, the cs can be set to a non-zero value to prevent scores of reaching near zero. After updating the smoothed scores, the observed score and move counters are reset.

$$s_m := \rho * \frac{s'_m}{mc_m} + (1 - \rho)s_m + cs \tag{11.2}$$

11.6. Simulated annealing temperature

As stated earlier, the simulated annealing algorithm keeps a temperature variable during execution. This temperature variable defines how tolerant the algorithm is in accepting worsening solutions. The temperature T is set to an initial temperature at the start of the algorithm. After each iteration, the temperature is multiplied by the temperature reduction parameter $\alpha < 1$.

When a move creates a new temporary solution x', the value of this move is compared to the current solution x. If the cost $f(x')$ of x' is smaller than or equal to the cost $f(x)$ of x, the temporary solution x' is always accepted. If $f(x') > f(x)$, then x' is accepted with a probability based on the difference and the temperature as given in Equation (11.3) [Ropke, 2005]. In the beginning of the algorithm worsening solutions are easily accepted, therefore exploring a varied region in the search space. As the iterations proceed, the temperature gradually lowers and the acceptance criteria is getting more selective. Where in the beginning it will accept even highly worsening solutions, later in the algorithm it will more often only accept slightly worsening solutions.

$$p = e^{\frac{f(x') - f(x)}{T}} \tag{11.3}$$

After some time, the temperature will be nearly zero and only improving solutions will be accepted. At this moment, it will usually soon get trapped in a local optimum, after which the algorithm stalls. When this happens, the temperature will be increased to T^0 again. This is called reheating. It enables the solution to diversify again and possibly reach a better optimum. The algorithm terminates after the maximum allowed amounts of reheats is reached.

11.7. Cost evaluation

During the execution of the algorithm, the costs will be calculated after evaluating every move, and for SA'' it will be calculated to evaluate the possibilities per move. This is done by first calculating the travelling times for the given trips, and then calculating the cost based on ship and project cost per duration. If a ship exceeds its capacity, a cost penalty will be added. The same holds if a turbine is installed by an incompatible method.

When the trip times are calculated, waiting times are introduce to fulfil the synchronization constraints. This means that if a ship is scheduled to deliver a component which has not arrived yet, it will wait at the harbour until it arrives. The same holds for other time synchronizations.

11.8. Discussion

The grasp algorithm currently evaluates all possible insertion points for a visit based on ship and trip location. As soon as the first visit for a turbine is decided, the corresponding method will be decided. If a single ship is part of multiple installation methods, one method will be picked randomly. This is done to decrease the amount of options to evaluate and thus the time required to run the grasp algorithm. However, it would be interesting to see what the effects are of evaluating all installation methods.

Furthermore, reheating now is done by simple resetting to the original temperature. However, by suddenly increasing the temperature by a large amount, the algorithm might move far away from the current solution. Although this diversifies the search space, it also means that certain work done on optimizing the solution is lost. It might improve the algorithm if more research was done for this balance. An example is not completely reheating, but slowly increasing the temperature. Another possibility is reheating instantly, but varying the magnitude of this.

Also, the penalties given based on ship capacities are now constant, increasing the cost if these capacities are exceeded. It might be interesting as well to research the effect of varying these capacities. At the end of the algorithm, the penalties have to be large enough to forbid any trips exceeding the capacities. However, during the execution of the algorithm this is not necessary. Varying the penalties might lead to temporarily leaving the solution space in order find better solutions later, although this principle is also present in the temperature acceptance criterion. Another possible approach would be to completely disregard penalties, and use moves which will never exceed the capacity penalty.

11.9. Conclusion

An algorithm is created with the goal of overcoming the problems encountered when using standard local search methods by introducing a nested second local search. Simulated annealing is used in order to overcome local optima and due to the probabilistic nature of both GRASP and simulated annealing, running the algorithm multiple times will result in exploring a larger part of the search space.

Since further problem-specific constraints can be implemented in the cost function, this algorithm is very versatile and is expected to handle changes in project requirements well. As long as the solution found can still be expressed as a collection of routes, changing the cost function can be enough to handle different projects. Penalties can be introduced to forbid certain outcomes and can be implemented independently of the solver algorithm.

The adaptive move selector provides automatic monitoring and implementation of move performance. Not only does this help to change the frequency of moves according to performance, this self regulating principle also makes the algorithm very expandable performance-wise. When new moves are added, little evaluation of their performance has to be done since the adaptive solver mechanism handles this.

Furthermore, the algorithm can be tuned for requested performance. A slow cooling algorithm with a lot of reheats will result in the best solutions and can be useful for the initial planning when a lot of time is available. During the execution of a project the algorithm can be tuned to run faster. This might result in solutions of worse quality, but allows the algorithm to be run in real time to adapt to changes during project execution. Finally by tuning to an even faster algorithm it can be used as a planning helper tool. It might be useful for example to quickly test what reducing or increasing the amount of plugs does. In this case the algorithm can be tuned to create very fast solutions, which give a quick impression of the consequences of certain decisions.

12

Results

As presented earlier, two solutions methods have been developed for each problem: an exact optimisation method and a heuristic. The roles for both methods are the same in each problem. For the exact optimisation method, this is to validate and evaluate the heuristic method. The exact optimisation only works for small instances, but by trying to get these instances as large as possible the heuristic can be validated with more confidence. The goal of the heuristic is then to solve the real-world problems.

The tests done are based on these purposes per model. Each exact method has as goal to improve the original MILP model. Therefore, the solver performance of this original model is compared to the proposed optimisation techniques. This gives an insight in the strength of these techniques. Subsequently, the heuristics are tested. The results of these tests are compared against the exact solutions. Afterwards tests will be done for the Walney project.

The tests were performed on the DAS-5 cluster of TU Delft and other institutes. Tests were done on a single node, with 8 dual cores which have a processor speed of 2.4 GHz. To compare this with a regular pc, a similar test which took 725 seconds in the DAS-5 cluster, took 800 seconds on a pc with 2 cores with a processor speed of 2.70 GHz. This change is thus not very large. All algorithms were created in Python. To solve the MILP problems, these were formulated in Python as well by using the PULP library (see Roy, J.S. [2017]). This library allows for a general way of formulating MILP problems, which can be linked to multiple solvers. Since the problems became quite big, expansions to this library where made in order to significantly decrease the time and memory needed to formulate problems. Besides this, extensions to include a warm-start were also written. After formulating these problems with PULP, they were solved with CPLEX.

To compare MILP solver performance, the term deterministic time is introduced. This stands for an arbitrary amount of calculations done. The benefit of using this in comparison with actual time, is that it is independent of other processes being carried out on the same computer. For example, a student compiling his latex document might slow down the solver running on the same computer. This technique is part of CPLEX Optimisation Studio and therefore not used in custom scripts.

Because deterministic time is a CPLEX custom measurement, the real value is not important and all times are normalised.

12.1. Computational Results Component Relocation Problem

A series of tests has been performed to test the optimisation methods for the component relocation problem. First, the performance of different cutting planes is evaluated. After this, the exact method is compared against the heuristic method and finally the algorithm is ran for the Walney project.

12.1.1. Problem configurations

For the component relocation problem, three types of flow configurations have been studied. The first configuration is called *loading*. This configuration handles the loading of a fixed set of components from a storage area to a ship. The initial field consists of an empty loadout area, and occupied and empty storage-only locations. On the occupied locations, the components are placed randomly. From there, they are requested to be loaded onto the ship, in ordered sets of batches of size 2. The loadout area therefore consists of 2 locations. All components are thus places on the storage area in the beginning and the problem is completed when all are unloaded.

The second configuration is called *storage*. This method combines arriving components with departing components. At the start of this problem, all locations are empty. The components arrive in batches of 2. After they all have arrived, departing starts in batches of 2 until all components are loaded onto the vessel. This problem thus exists of first unloading all components to the storage area, before loading them all to the ships.

Finally, the configuration called *buffer* is introduced. Where the first problems both modelled a project where the components were all available at the start of the project, the buffer configuration models a mixed flow of arriving and departing components. It is then per case specified what the in- and outflow of components is, as well as the field geometry. In all cases it holds that, except for the starting and ending conditions, the storage area will never be empty.

12.1.2. Cut Performance

In the first batch of tests, the influence of cutting planes in the CRP is evaluated. Multiple instances per problem configuration are solved by different methods, and the performance of these methods are compared against each other. First, Algorithm 9 is ran for each instance to get a heuristic solution. The amount of moves in this solution is used as an upper bound, such that the variables x_{tkij} can be initialised.

In Chapter 7, multiple cutting planes were introduced. By ignoring the spatial blocking constraints, a lower bound could be calculated based on the flow of components and the available loadout space. This is defined as the *min-moves* cutting planes set. Furthermore, the *double-relocate* cutting planes set forbade moving the same element in consecutive timesteps in the storage-only area. The third cutting planes set is called *time-symmetry*, which removes symmetrical solutions by shifting all moves to the beginning of the available time. Finally, a method is introduced called *all*, which includes all cutting planes. These cuts are summarised in Table 12.1

	Description	Equations
Min-moves	Minimum moves without considering blocking constraints	(7.4)
Double-relocate	Minimises redundant direct relocations of one blocking	(7.7)
Time-symmetry	Shift all moves to beginning	(7.6)
All	Combines all cuts	(7.4),(7.6) and (7.7)

Table 12.1: CRP cutting planes

For each problem configuration, multiple instances were solved by all cutting planes configurations. The results are shown in Appendix D.1.1. A time limit of 2 hours was set for all problems. In Table 12.2, the percentage of problems solved per cutting-plane configuration is shown. Each configuration improves the amount of problems solved, however it is interesting to see that combining them did not improve the percentage of solved instances. It can also been seen that blocking consecutive relocations of the same component leads to the highest improvement.

	Det. time			Nodes		
	Buffer	Loading	Storage	Buffer	Loading	Storage
None	1.0	1.0	1.0	1.0	1.0	1.0
All	2.0	0.7	0.8	3.6	0.4	0.6
Double-relocate	1.1	4.8	3.9	1.3	13.2	8.0
Min-moves	1.7	1.3	2.0	13.6	0.9	3.3
Time-symmetry	2.2	1.3	2.1	4.7	0.8	1.9

Table 12.3: Time and nodes for crane optimisation performance

	Solved
None	83.3%
All	83.3%
Double-relocate	94.4%
Min-moves	88.9%
Time-symmetry	88.9%

Table 12.2: Exact instances solved

To investigate the performance, only instances have been selected which have been solved for all cutting-plane configurations. Both the deterministic time and explored nodes are evaluated per problem and cutting-planes configuration. The total time and nodes required for solving all problems, is shown normalised with respect to the performance without additional cutting planes in Table 12.3. From these results, it can be concluded that not all cuts increase the performance of the exact solving method for these instances.

Time-symmetry and *min-moves* do not improve the performance on nodes nor time. Although double-relocate improves the performance for the combined configuration, it significantly increases the time and nodes on other problem configurations. However, it can be seen that for the storage and loading problem configurations, adding all cuts improves the performance. Besides this, it seems that the value of the cutting planes will only become apparent for more difficult problems. In Table 12.3 only instances have been considered that were solved for all cutting-plane sets. This was necessary for comparison. However, this meant that more difficult problems were not taken into account.

The rolling horizon algorithm solves the subproblem by dividing it into even smaller problems. To initialise the x variables, a maximum time has to be set for which the problem can be solved. Currently, this is done by calculating a lower amount of moves and incrementing this by two. If no solution will be found, the solver is ran again while increasing the amount of possible timesteps.

When all of these smaller problems are solved, the solutions can be combined to create a (sub)optimal solution for the subproblem. This is used as a warm start, and the solver will be ran to find a better solution.

Therefore, two things are interesting while considering the starting conditions of the algorithm. By introducing the extra time, the amount of available timesteps is usually higher than the amount of moves in the initial condition. To evaluate the effect of these extra timesteps, each solved problem-instance is ran again for the starting condition *tight-upperbound*. This starting condition sets the total amount of timesteps available equal to the amount of moves in the optimal solution.

Furthermore, since the RH algorithm uses a warm start, the effect of this is evaluated as well. The starting condition named *warm-start* solves each problem, while using a sub-optimal warm start, generated by Algorithm 9.

	Det. time	Nodes	Solved
Standard	1.00	1.00	86.7%
Tight-upperbound	0.92	0.98	86.7%
Warm-start	0.98	0.81	90.0%

Table 12.4: Performance with different starting conditions

The normalised results of these tests have been summarised in Table 12.4. It can be seen that using this additional information improves the performance on all criteria. The amount of nodes

was decreased with 19% by using a warm start, which is a significant improvement. In the next section the performance of the rolling horizon algorithm will be tested and the results will be evaluated.

12.1.3. Heuristic performance

To evaluate the heuristic performance, tests were done for all instances in Section 12.1.1. This was done for multiple settings. The setting set *standard* allows blocking components and puts a penalty of 1 for each blocking component. The penalty set for unavailable spaces is equal to 0.2. The motivation for this number is that it will not likely force extra moves to minimise unavailable spaces, but given two equivalent moves the one is chosen which results in the lowest amount. Furthermore, the cost of a component on storage-only locations at the end of the rolling horizon time window is set to 1.

Another configuration of settings is introduced called *double blocking*. As its name implies, this configuration sets penalises blocking with a cost of 2. The reasoning behind this is that 1 is a lower bound for the amount of moves resulting from a blocking component, but there is a possibility it will result in more than this. Therefore, tests are done with a double costs. Furthermore, a configuration is created with the name *loadout*. This configuration sets the cost for locating components on storage only to 0.99. The reasoning behind this is to avoid it if possible, but not waste any extra moves on it. Therefore, components will be more often left at loadout locations. It must be noted that this decreases the lower bound given in Equation (7.5). Additionally, the configurations sets *no-blocking* and *cluster* are introduced. Both configuration do not allow blocking at the end of the horizon step, and the latter also includes the clustering variables from Equation (8.7).

For all tests, both the exact solver as the heuristic algorithm was ran, and the results were subsequently compared. These tests are shown in Appendix D.1.2, and show that for instances with up to 46 moves in the optimal solution, all heuristic results were found equal to this exact solution. The only instance where a significant difference was noted between the different configurations was for a combined in and outflow consisting of 18 components. Since the algorithm was terminated if a step could not be solved within 3 hours, all solvers except for the setting clustering did not came to a result. The cluster setting, however, came to the exact solution.

12.2. Operational results crane optimisation

All methods for the heuristic optimisation were used on the walney arrival schedule with a horizon of 2. The duration of executing these algorithms was very large, taking nearly a week to finish. It was however shown that the results obtained from this are significantly better than the manually made schedule. The exact moves of the best solution can be found in Appendix D.1.3. The amount of crane movements done is 336. In Section 3.5 it was shown that in the crane movement schedule 363 moves were accounted for, of which 189 were mandatory vessel loadout moves. When using the lower-bound algorithm of Chapter 7, a lower bound of 314 moves was found.

	Scheduled	Algorithm
Moves	363	336
Lower bound moves	314	314
Upper bound additional moves	49	22

Table 12.5: Operational results crane movements

As summarised in Table 12.5, this means that in the crane schedule included a maximum of 49 more moves than required. The generated schedule has at most 22 additional moves, 27 less than the schedule. This means that there is at least a decrease of 55% in unnecessary crane movements.

12.3. Computational results Ship Installation Routing Problem

12.3.1. Problem configurations

The tests for the SIRP are carried out on three different fields. The first field is the *Walney field*. The *Walney field* has a rather standard grid-like layout. Besides this, two other layouts were considered. A recent research development in the wind industry is to optimise the layout of turbine fields in regards to wakes created by other turbine. This is a promising development, but it leads to less grid-like layouts where thus intrafield routing has an increased impact. Therefore, it is also interesting to see how the solution method performs on these fields. For this reason, the coordinates were taken from the real-world Hornsrev field, based on an illustration in Fischetti and Pisinger [2017] and from an artificial case, generated by the model in Fischetti and Monaci [2016]. This third field is referred to as *artificial field*.

Since, as in the CRP, the SIRP for these fields is too large to solve exactly, multiple tests are done for subsets of all turbines per field. These are selected by choosing one random turbine and selecting the closest n turbines from this. This is done because it is expected that in an optimal solution, routes will usually go from a turbine to a close neighbour. Since no plugs or installation method requirements are available for the *Hornsrev field* and the *artificial field*, these are set random for all three fields. Furthermore, to adapt to the smaller field instances the capacity for the Aeolus floating installation method is changed from 5 to 4. The costs per ship and for project duration are all set equal.

The tests are done while considering four types of tasks. The first tasks, called *free-installation*, considers only the requirements that each turbine has to be installed by a compatible installation method, but without any temporal constraints. This means that two versions of the Aeolus, floating and jacked, can be present in the field at the same time. It also means that A transition piece can be installed before an monopile. Although this is not applicable to the Walney project, it might be useful for other cases. Moreover, it serves as a comparison for the other tasks.

The second task-set models just the Svanen and the Aeolus without any plugs. Temporal constraints are taken into account here. This task set is called *precedence*. Additionally, the task-set called *plugs* adds the buoyancy plugs to this model and finally the task set *arrival* also adds delayed arrival of transition pieces.

12.3.2. Exact performance

To get an indication of the performance of the exact optimisation model, first the model without any cuts is ran. The runtime limit was set to four hours and tests were ran for 6 turbines. The full results are shown in Appendix Appendix D.2.2. In Table 12.6, the results are summarised. Although integer solutions were found for all instances, optimal solutions were found only for the precedence class. For all other instances there is at least a 10% gap between the lower bound and the best solution found after the time limit is exceeded.

	solved	gap
blank	0 %	19.8 %
plugs	0 %	17.2 %
prec	100 %	0 %
arrival	0 %	12.0 %

Table 12.6: Results without any cuts

It might seen counter-intuitive that the precedence test instances are solved to optimality and not the blank test instances. Upon further inspection of the results it is shown that the linear relaxation value is 0 for all blank test-instances and 10 for all other test-configurations. This is a consequence of the time offset in the ship precedence constraint Equation (9.13d). The model assumes that the transition period happens regardless if turbines are installed by the second ship of the precedence relationship. This imposes a lower bound which might be the cause of the precedence-tests solving to optimality while the blank-tests don't.

However, besides the instances for the precedence relationship, no optimal solutions were found for the case of 6 turbines and the solver performance on the original MILP model Chapter 9 can be considered as quite bad. For this reason, multiple improvements were proposed in Chapter 10, and in the remainder of this section those improvements will be evaluated.

For this purpose, three evaluations are done. First, multiple cuts are added to the model individually. It will show that certain cuts improve the performance significantly. Some of these cuts will be used as standard cuts in the model. Subsequently, the remaining cuts are tested again, but now combined with these standard cuts. Based on this it is decided which cuts improve the performance. Finally the Branch and Bound method is evaluated.

The tests were done by adding multiple cutting-plane configurations from Chapter 10. Three of these did not including extra variables. The *total-time* constraints sets the makespans larger than or equal to the length of the routes. *Trip-symmetry* removes symmetric solutions by shifting all routes the beginning of the available route positions, and the *field-trip* cutting plane configuration puts a lower bound on the trips to the installation field. Besides these, 4 more cutting plane configurations were introduced in Chapter 10 which use additional variables. The flow based sets introduce additional variables to define the flow per arc, and the waiting-based sets introduce additional variables to define the waiting time per nodes. These are listed in Table 12.7

	Description	Equation
Total-time	Makespan time larger than cost of routes	(10.1),(10.2)
Route-symmetry	Only use route if preceding route is used	(10.3)
Waiting-arc	Variable to define waiting time on arcs	(10.7)
Waiting-node	Variables to define waiting time on nodes	(10.9)
K-flow	Define flow per arc per ship	(10.13)
Kr-flow	Define flow per arc per route per ship	(10.14)
Field-trips	Lower bound on trips to field	(10.5)

Table 12.7: Cutting-plane sets for the SIRP

Single cuts

In the first batch of tests, the individual performance of the cutting-plane sets is evaluated. For the multiple test instances, the problem is solved to optimality by each set of cutting planes. The full results are available in Appendix D.2.3. Here, it can be seen that only problems from problem types *blank* and *precedence* are solved to optimality for all cutting planes. The normalised time and nodes used for these instances are shown in Table 12.8.

It can be seen that every cutting-plane configuration results in a decrease in both time needed and Branch and Bound nodes explored. By far the largest improvement is due to the *total-time* cutting plane set, which has improves the formulation without additional cutting planes by over 99%. Furthermore, it can be seen that *route-symmetry* and both flow formulations are the other cutting planes with a very large performance improvements. The difference with *k-flow* and *kr-flow* is not very clear, as *k-flow* requires less time, but *kr-flow* requires less nodes.

	Time	Nodes
None	1.000	1.000
Demand	0.887	0.750
K-flow	0.285	0.396
Kr-flow	0.339	0.388
Total-time	0.003	0.005
Trip-symmetry	0.284	0.334
Trips-back	0.842	0.872
Waiting-arc	0.925	0.884
Waiting-node	0.909	0.803

Table 12.8: Normalised result with single cuts for problem types *blank* and *precedence*

To analyse the problem configurations arrival and plugs, the amount of instances solved was evaluated in Table 12.9. For clarity, only improving cut-sets are considered here. Trips-back and waiting-nodes solved the same instances as the formulation without additional cutting planes. It is interesting to see that the *min-routes* cutting-plane set solves less instances for the plug-configurations, but solves all arrival-instances. Furthermore, it is shown that *kr-flow* is the only configuration which solves all instances.

Based on these results, and on additional small tests, the cutting planes *time-symmetry*, *total-time* and *field-trips* are considered standard cuts which will be included in every exact solution method. Another motivation for excluding other the cutting planes from the standard set, was that those include additional variables. As was shown in small extra tests, that this influences the solver performance often in a less straightforward way, and therefore require extra evaluation.

	Plugs	Arrival	Total
None	67	33	50
Kr-flow	100	100	100
K-flow	100	67	83
Total-time	100	67	83
Waiting-arc	100	67	83
Min-routes	33	100	67
Trip-symmetry	67	67	67

Table 12.9: Amount of instances solved for plugs and arrival (Single cuts)

Combined cuts

Based on previous tests, the standard cut set was defined as *route-symmetry*, *total-time* and *trips back*. It is expected that, especially for larger problems, these cuts will improve the performance, regardless of which other cutting planes are added to the model. For cuts which contain additional variables, more testing is done in this section, since adding new variables might result in unexpected behaviour. Therefore, a new batch of test is done where the non-standard cuts are evaluated. Each problem instance is solved once with the standard cut-set, and once for each non-standard cutting-plane set from Table 12.7, in combination with the standard set.

	Time	Nodes
Standard	1.00	1.00
K-flow	0.33	0.24
Kr-flow	0.40	0.28

Table 12.10: Results for combined cuts for instances solved for every cutting plane set

In Appendix D.2.4, the results of these tests are given. From these results, the problem instances were taken which were solved by all cutting-plane sets. By taking the average nodes and time needed per cutting-plane set, it was shown that only the flow-based cutting-plane sets further improve the standard configuration. The result however, was significant, and *k-flow* shows to improve the standard cutting-plane set the most by both nodes and time. Both flow formulations were also capable of solving all instances.

More information about the results is given in Table 12.11. This shows quite favourable results. For the most complete problem type, *arrival*, the improvement by k-flow is almost 90%. It also shows that for each problem type the amount of nodes is reduced, although for the precedence case the time needed is increased. It can also be seen that, except for the blank problem-type, *k-flow* performs better than or equal to *kr-flow* on both nodes and times.

Problem type	Cutting planes	Time	Nodes
Blank	**Standard**	1.00	1.00
	Kr-flow	0.20	0.04
	K-flow	0.32	0.06
Plugs	**Standard**	1.00	1.00
	Kr-flow	0.78	0.51
	K-flow	0.73	0.48
Precedence	**Standard**	1.00	1.00
	Kr-flow	1.79	0.75
	K-flow	1.32	0.75
Arrival	**Standard**	1.00	1.00
	Kr-flow	0.32	0.18
	K-flow	0.20	0.12

Table 12.11: Normalised average time and BB-nodes required per problem type

Branch and Cut

So far, the exact tests use the standard Branch and Bound algorithm of CPLEX. In this section, test results are presented for the Branch and Cut algorithm. Although this algorithm uses CPLEX as well, there are some implementation notes which affect the way of testing. Because of this, first the implementations issues will be discusses before presenting the results.

The Branch and Cut algorithm is created by adding callbacks to the Cplex Branch and Bound algorithm. A callback is a custom function which will be executed at certain steps in the algorithm. By registering a callback after branching node optimisation, custom code is called each time that a branching node produces an optimal (non-integer) solution. In this custom code, new constraints are generated and added to the problem.

There are two drawbacks of this method regarding implementation. First, the custom code is many times slower than CPLEX. The custom code is written in Python, while CPLEX is written in C, a significantly faster programming language. Besides pure performance of code there is also overhead since variables have to be converted from C to Python, and constraints have to be converted the other way around.

Secondly, normally CPLEX uses multiple cores to run the optimisation. This means that the work is split in parts, which are divided amongst computer cores. Letting code run in multi-core is not a trivial task. It needs to be defined how code is split amongst the cores, and how the cores communicate. Since this was not implemented in the custom code, multi-core processing is turned of during the Branch and Cut algorithm.

Two branch-cut methods have been introduced in Chapter 10. *Branch-cut-1* defines multiple ship-sets of which at least one has to visit each turbine, and adds constraints based on this. *Branch-cut-2* defines the demand per turbine based on the installation method variables. For each method, two cutting-plane sets are added, one in combination with the *standard* cutting planes set and once in combination *k-flow*. The cutting plane sets are listed in Table 12.12. 1e−10

	Description	Additional equations
Standard	Standard set	
K-flow	Branch-cut-2 with k-flow	(10.13)
Branch-cut-1	Branch cut based on ship sets	(10.19)
Branch-cut-2	Branch cut based on installation method	(10.24)
Branch-cut-1k	Branch-cut-1 with k-flow	(10.19),(10.13)
Branch-cut-2k	Branch-cut-2 with k-flow	(10.24),(10.13)

Table 12.12: Cutting plane sets for branch-cut, all include the standard-set

For these tests, initially the performance is evaluated for instances which were solved for all cutting-sets. From these instances, the cutting-plane sets which had improvements on either time or nodes are shown in Table 12.13. It is shown that all Branch and Cut methods improve the standard formulation, but only *branch-cut-2* improves *k-flow* on both nodes and time required. It is also shown that adding the *k-flow* cuts to the *branch-cut* cutting-plane sets worsen the performance. When looking more closely at the results in Appendix D.2.5, it can be seen that for these instances, the amount of cuts added by the BC-algorithm is nearly zero. It thus seems that the *k-flow* cutting planes almost completely force the demand per node-subset to be met at the non-integer solutions. This prevents branch-cut constraints from being added.

	Time	Nodes
Standard	1.00	1.00
K-flow	0.19	0.08
Branch-cut-1	0.20	0.14
Branch-cut-1k	0.19	0.08
Branch-cut-2	0.03	0.02
Branch-cut-2k	0.18	0.08

Table 12.13: Performance branch cut

The same tests can be viewed per problem type, as shown in Table 12.14. For all but the blank problem-type the results are consistent with Table 12.13. For the *blank* problem-type *branch-cut-1* actually performs better than *branch-cut-2*, although *k-flow* performs even better here. Besides this, the deterministic times of the callbacks were evaluated. Due to the complexity of the Edmunds-Karp algorithm, it was not expected that the time to solve the separation problem was significant. In Appendix D.2.5 it is shown that the highest percentage of callback time per total time is 1.75%. When overhead (writing variables from C++ to Python) is ignored, this percentage reduces to 0.0143%.

	Blank	Precedence	Plugs	Arrival
Standard	1.000	1.000	1.000	1.000
K-flow	1.849	0.011	0.433	0.020
Branch-cut-1	2.098	0.012	0.287	0.092
Branch-cut-1k	1.849	0.011	0.433	0.020
Branch-cut-2	0.258	0.035	0.168	0.006
Branch-cut-2k	0.866	0.011	0.858	0.020

Table 12.14: Nodes branch cut per problem

12.3.3. Heuristic performance

To evaluate the performance of the AISA algorithm, multiple small tests are done to compare the exact solution against the heuristic solutions. The tests were done for small field consisting of 6 turbines. The tests were done with two different settings for the repair move. The setting *repair-broken* uses a repair-move where only the necessary visits are removed and re-added in order to create feasible routes. The tests with the *repair-waiting* settings removed these visits, together with all visits where a ship was waiting. Each test included three GRASP-restarts, and the best result of these was selected. The full results of these tests are available in Appendix D.2.6.

It was shown that for the problem configurations *blank* and *precedence*, all tests came to the optimal conclusion. For the other two configurations, in both cases 2 out of 3 tests came to an optimal solution for both settings. For these configurations, the average gaps between the optimal and exact solutions are shown in Table 12.15. The other two problem types are not shown here since the gaps are zero. It can be concluded that introducing plugs in the problem decreases the simulated annealing performance. The repair-waiting settings shows better performance in both cases, especially in the full problem which also considers arrival times of components.

When comparing the durations of both algorithms, the repair-waiting method took on average 7.2% percent longer to finish in comparison with the repair-broken method.

	Repair-broken	Repair-waiting
Plugs	0.53%	0.28%
Arrival	0.87%	0.06%

Table 12.15: Gap between heuristic solution and exact solution

12.4. Operation results Routing optimisation

For further testing the simulated annealing algorithm is ran multiple times on the full Walney configuration. Two problem types were handled, the problem with component arrival times and the problem without component arrival times. Both types are ran for the repair-broken and the repair-waiting setting. Since for the size of the Walney project (87) turbines the broken-waiting algorithm took significantly longer, this algorithm was ran with a maximum of 1000 iterations and the algorithm with the repair-broken settings was ran with a maximum of 6000 iterations. The best route is given in Appendix D.2.7 and Appendix D.2.8.

Comparison of the results was done against the original Walney schedule. It was discussed in Chapter 2 that this original schedule detail in intrafield travelling and plug waiting time. For this reason the original routes where considered, but evaluated against the same distance-matrix c and costs as the heuristic tests. Since the planned Walney schedule was created with different (constant) sailing times and without considering plugs, adding synchronization and component arrival times would result in a really bad result. Therefore, this is ignored in the cost-calculation based on the original schedule.

In Table 12.16 the resulting costs are shown. It shows that when component arrival times are included, the solution found by the heuristic is increased compared to the Walney Schedule, but that without arriving components the cost is significantly decreased.

	Cost (€)	Normalised
Scheduled	607.288	1.00
No arrival heuristic	553.717	0.91
Arrival heuristic	735.002	1.21

Table 12.16: Costs for best solution per problem setting

The average intrafield distances travelled (per trip between two turbines) are shown in Table 12.17. This show that for both the Aeolus floating as the Svanen there is an increase in sailing time, and for the Aeolus jacked there is a significant decrease. When taking the average over all trips between turbines, it is seen that the intrafield sailing time increases by 4% and 17% for without and with component arrival respectively. Comparing this to Table 3.2, this increase is significantly lower than the difference between scheduled and executed routing.

The average time for these tests was around 21 hours. When evaluating the plug synchronization, it was found that waiting on plugs occurred twice, totalling to 8 hours. This is a very large improvement compared to the 96 hours in the project execution.

	Aeolus floating	Aeolus jacked	Svanen	All	Increase
Arrival heuristic	1.82	1.97	2.58	2.15	17 %
No arrival heuristic	1.71	1.75	2.24	1.92	4 %
Scheduled	1.43	3.66	1.75	1.84	0%

Table 12.17: Average sailing distances (km)

12.5. Discussion

It was shown that the costs are reduced by the AISA algorithm for the SIRP. However, it is important to place this conclusion into context. An offshore wind turbine farm installation project is an elaborate project including many aspects. A full time spends many hours on discussing the details to create a well suited planning.

Even with the most important aspects taken into account, there are still factors not included in the model. An example is contractual obligations to finish a subset of all turbines before a certain date. This was included in the Walney project, together with multiple other non-modelled, project specific requirements.

Therefore, simply looking at the costs saving and drawing conclusions might be considered naive. The performance of the algorithm can better be evaluated by looking deeper into the generated schedules, and comparing the sailing distances and synchronization problems

12.6. Conclusion

For the routing optimisation, it was shown that the introduction of plugs in the problem decreased both the exact as the heuristic solver performance. The MILP formulation from Chapter 9 performed very bad, but significant increases were made by adding the standard-set of cuts. For further performance improvement the branch-cut-2 method from Section 4.5.1 has

theoretically the best results, although if implementation is considered as well k-flow can be seen as the most practical method since there is no need to modify the solver.

For the AISA it was shown that the repair-waiting method where all nodes with bad synchronization were removed and re-added to the solution was slower, but amounted to better results. Upon optimisation of the Walney project it was revealed that the solver did not increase the total cost and that the installation methods chosen were quite different than the planned order.

However, upon further inspection it was shown that although the average distance travelled between nodes was higher in the optimised solution than in the original schedule, this increase was significantly lower than the increase from the scheduled to the execution route. It was also shown that the amount of plug synchronization issues was very low.

For the crane movements it was shown that the performance due to the added cutting planes did not always improve the speed of the solutions. However, when considering the amount of instances solved the cuts did improve this. This suggests that the strength of those cuts becomes mostly apparent in larger and complicated problems.

The rolling horizon algorithms produced solutions of high quality, as for every problem instance for which a lower bound was found the rolling horizon produced at least one solution with this amount of moves. It was also shown that adding a clustering variable increased the amount of solvable problems. A possible explanation for this is that the clustering variable prevents the algorithm to enter state where most empty locations are blocked.

Furthermore, the operational results of the CRP resulted in an improvement of at least 55% based on the non-necessary crane movements. This solution took over 6 days to calculate, but the maximum amount of calculating a single step was only 12 hours. For the simulated annealing algorithm the solution was created in 10 hours.

Comparing these times to the times of ship trips, it can be concluded that a choice can be made in less time than it takes a ship to complete its trip. Since the rolling horizon algorithm improved the original schedule, it can be concluded that this algorithm can be implemented directly in future projects with similar storage requirements. For the simulated annealing algorithm, even although it did not improve the original schedule, the choices made during the executed resulted in a larger increase of travelling time. On top of this, the simulated annealing algorithm showed good synchronization results and thus it can be used as a decision support tool during wind farm installation projects.

13

Recommendations, Discussion and Conclusion

13.1. Recommendations

Based on this thesis, some recommendations are given for future research. When considering what is done it can be seen that two separate problems were solved. The next step would be to combine these two, finding a schedule which optimises both. The AISA already contains an option for delayed arrival of components, but not for costs based on relocations. In theory the two presented heuristic algorithms could be linked together, calculating for each iteration in the simulated annealing algorithm the optimal amount of moves and including this in the cost. Unfortunately, this would take an immense amount of time to solve and realistically cannot be done.

However, in combined optimization it is not required to know what exactly the TP movements are, but instead it is enough to know the amount of steps. It also might be possible to work with an indication of this amount instead of the exact amount. This might be done in a similar approach as the lower bound algorithms presented, with some addition for an estimate on the amount of blocking components based on in and outflow. However, before doing this it must be noted that the relocation prices are almost insignificant compared to the price of installation ships and research has to be done if it would in any case be profitable to change the ship routing in favor of crane optimisation.

Furthermore, it must be noted that the goal of the algorithms is to provide more flexibility and faster decision making during the execution of projects. To evaluate this further, a modelling assignment should be done to model how the optimisation would behave when used. In this model an initial planning should be made for an installation project. A simulation will then be done based on the decisions of this planning, but with more random variables, including varying weather and installation circumstances. Periodically during the simulation, feedback will be given to the optimiser, allowing for changes in for example sailing times and a new planning should be made based on the remaining tasks. This will evaluate how the optimisation methods deal with the randomness present during offshore installation projects.

If it turns out that the model does not handle this randomness very well, it is recommended to look into research for randomised vehicle routing. Since both sailing times and installation times are modelled as arc costs, a VRP model with stochastic arc costs would cover both aspects. Han et al. [2014] researched the vehicle routing problem with only rough stochastic information about arc costs in the future, which seems as a good starting point for what is possible for this.

It is also noted that the travel times for the Walney project are based on the distance between turbines and the average intrafield velocity. In reality this assumption overestimates

the traveling time on longer trips since ships will reach a higher velocity here. For implementation in real projects this should be considered. This does not call for any changes to the model, but the cost matrix should be calculated while taking into account the increased traveling speed on long distances.

On a computational level it was shown that the VRP with synchronization and multiple installation methods is a very difficult problem. Even with multiple cuts added the performance is nowhere near the performance of the exact methods for the traditional VRP. Based on literature and tests, Branch and Cut is proposed as the most promising direction for further research. There is a broad selection of cuts available for this. A few of these were converted to work with multiple installation methods, and even while they lost some strength in the process, it is recommended to try a similar approach for other cuts. This could be, for example, the multistar cut-family as found in Letchford et al. [2002].

With the addition of new branch-and-cut cutting planes and by handling larger problems, the amount of constraints added might become too large. Currently, the cuts are added based on the total supply to a subset of nodes. If this supply is not enough to satisfy the demand of that subset, a constraint will be added. To decrease the amount of cuts added, it might be possible to add threshold there. For example, only add cuts if the total supply is 80% of the demand. It should be tested what the effects of this are. If this approach falls short, then it is recommended to introduce cut management. In this approach a cut-pool is maintained where cuts are not only added to the problem, but also removed. A good starting point for this might be the research of Lysgaard et al. [2004], who presents a branch-and-cut algorithm for the CVRP with cut management.

Similarly, more research should be done to increase solving time for the CRP MILP model. This is especially important since any improvements will also improve the rolling horizon heuristic. To do this the recommended direction is to look on how to decrease the problem size. The reason for this is that the rolling horizon heuristic slows down significantly on steps with a lot of components loaded. This is expected as it has more components to consider for relocations. Disallowing certain components to move components might lead to worse solutions, but since a significant decrease of computational time is expected this is recommended for further research.

A similar approach could be implemented in the direct unloading algorithm. The direct placement algorithm for the CRP first determines the optimal locations for the arriving components, and then an algorithm is applied to see if it is possible to directly relocate to these positions. This is done while taking into account the blocking constraints. Although this works well and decreases the solver-time, it is very problem specific and changes in the neighbour-based blocking constraints might require a very different algorithm. It is therefore recommended to investigate the performance of using an MILP model for this problem. This MILP model would be a variation of the CRP model given in this thesis. It would require a maximum of one move per arriving component and no possibility of movements for other components besides the arriving ones.

Furthermore, the simulated annealing algorithm showed good results, but showed difficulties when the time synchronizations were very tight. For this reason it is recommended to look into a different algorithm for the inner-optimisation in the iterated local search. The reason for this is that a part of the subproblem, namely the plugs, is very constrained and simulated annealing does not make use of this property very well. The reasoning is that when all ship routes are set, the corresponding plug routing problem has a relatively small search space. It was noted that most work on deciding the plug routes was done by the repair phase rather than the second simulated annealing. It is therefore recommended to look into techniques as constraint programming or even MILP optimisation.

Finally, it also must be noted that the penalty inner step in the AISA slows the algorithm significantly. Although this part is definitely required, no tests were performed to see the effects of only running this second step periodically, so for example once in every 10 top iterations. This would increase the amount of top-iterations done. .

13.2. Discussion

In this chapter some discussion points will be presented. Firstly the modelling part is discussed. To make optimisation possible multiple assumptions were done and the problem was modelled as a linear model. One can question how valid an exact model is for a process like wind farm installation which is subjected to much randomness. However, it should also be noted that in the current planning a lot of random variables are fixed as well, or simply accounted for by using a large buffer.

Another possible issue in this thesis is how valid the comparison of the model with the planning is. In this planning, plugs were not considered as a possible bottleneck, and all intrafield routing times are set to constant values. Furthermore, the installation times assumed in the original planning are taken much larger to account for possible bad weather. These concerns were mitigated as much as possible by evaluating the chosen route by the same travelling times as used in the optimisation, but it is certainly a discussion point how valid it is to judge a schedule while changing certain assumptions. It remains not fully clear if the improvement from the simulated annealing algorithm is due to its own quality or due to having access to more information.

It is also an issue that there are, and always will be, unmodelled things which might influence the planning process. An example of the Walney project are contractual details which require deadlines on subsets of certain installation notes. This causes problems for two reasons. Firstly, again during the comparison with the real Walney planning, the question can be made if any improvement is created due to the strength of the algorithm or due to the fact that certain requirements are made. Secondly, it causes problems for using the algorithm since the suggestions made by it might interfere with these details.

Another issue is how valid are the comparisons done for the simulated annealing algorithm with the lower bounds. It showed good results and achieved optimality in multiple cases, but due to the weak MILP formulations this was only possible for very small cases. It must be noted however that even for small instances it is valuable to have validation. Firstly because it is a sign that an optimum can be reached, but moreover it can be used as a validation to determine whether the algorithms used in for example cost evaluation are correct. For a certain route, the waiting times have to be calculated at each turbine before calculating the turbines. This is not a trivial task since waiting at one position might cause a requirement for other ships to wait as well. Furthermore, in the optimal solutions ship might start later to decrease makespan cost. Things like these are very easy to mis during the construction of a simulated annealing algorithm.

Finally it can be questioned whether certain options in the model could not easily be decided manually. An example of this is the amount of turbines installed per installation method. This is a very defining factor in project duration and not very difficult to calculate. It might unnecessary to leave this decision to be made by the optimisation method. On the other hands, the methods presented are easily adaptable to implement a fixed choice here.

13.3. Conclusion

In this thesis optimisation methods are given for two different problems occuring during offshore wind farm installation, namely the Ship Installation Routing Problem and the Component Relocation Problem. By evaluating the decisions made in the planning and execution of the Walney wind farm installation projects the conclusion was made that it is commercially interesting to look into these problems. It was also concluded that there was a need for an adaptive method since it turned out that the actual execution deviates from the planned schedule, and that this deviation caused a delay and increase of cost. Furthermore, the details which were not considered bottlenecks during the scheduling phase turned out to have significant effect on the execution of this project.

The intrafield routing problem was modelled as a Vehicle Routing Problem. The installation field and harbour were represented by a network with nodes for each turbine and harbour locations and arcs between them. The optimisation method then looks for a way

to route the ships over these arcs while setting the node visiting requirements based on the installation methods. An MILP model was developed for this and several cutting planes and additional variables were explored. The most promising method was Branch and Cut. Although the performance was limited by implementation issues caused by Cplex, this method showed an improvement based on nodes explored and time needed.

It is further concluded that the combination of optimisation by makespan in combination with synchronization constraints weaken the LP formulation and additional cuts are needed to get a non-zero LP lower bound. Furthermore, the synchronization requirements are not compatible with any Branch and Price formulation found so far.

To solve real-world size problems an iterative simulated annealing algorithm was developed. An Adaptive iterative simulated annealing algorithm was developed which temporarily leaves the search space after a move, and the repairs an re-optimises the solution to return to the search space. This was done in order to prevent the synchronization constraints from invalidating or worsening the neighbourhood of a solution.

Compared to a real-world installation schedule showed a decrease in cost but an increase in travelling distance. However, a similar increase was also found when comparing the planning with the execution of the project, and the increase by the simulated annealing algorithm was significantly larger. It was also shown that the algorithm was able to significantly decrease synchronization issues between vehicles. Therefore, it is concluded that the AISA is well suited for the SIRP.

An MILP model for the CRP was created as well and a rolling horizon algorithm was created based on this, which divides the problem into chronological overlapping subproblems which are solved one by one. The optimisation of these subproblems was done both on amount of crane movements and layout quality. It was shown that stimulating the solver to put components next to each other, while preventing components to block each other based on departure time created the best results. It was also shown that using simple algorithms to define a lower bound and providing an initial solution to the MILP solver improved the amount of problems solved within a set time limit.

The solutions found by the rolling horizon algorithm were of high quality, both compared to optimal solutions and compared to the crane schedule for the Walney project. The most influencing factors on the duration of the algorithm are the size of the storage area and the amount of components passing through. To optimise the crane movements for a whole wind farm project several days are needed to compute a solution. However, the rolling horizon algorithm is capable of calculating a partial solution for moves in the near future without computing the whole algorithm. A single shipload can be calculated in several hours, which is in comparison to the time in between ship arrivals more than enough. It is therefore concluded that the rolling horizon algorithm is a very good fit for crane optimisation problems during wind farm installation projects.

It is thus concluded that both algorithms can provide valuable information to the decision maker during the execution of a wind farm project in the time available. In contrast to manual planning, new information can be processed quickly and changes can be made to the original planning. On an operational level it is therefore believed that the work done in this thesis will help decrease cost and increase efficiency during the installation of offshore wind farms.

From operational research viewpoint, a vehicle routing problem is introduced which combines the aspect of multiple split delivery methods with multiple time synchronization constraints. Multiple delivery methods in this form were not found yet in literature, but are crucial in using the VRP for installation projects.

Additionally, in container relocation problems, no MILP was found which included both unloading, loading and pre-marshalling. The constraints introduced in this thesis can be directly implemented in the model of Caserta et al. [2012].

Nomenclature

Abbreviations

SA'	Upper SA in Adaptive Iterative Simulated Annealing Algorithm
SA''	Upper SA in Adaptive Iterative Simulated Annealing Algorithm
AISA	Adaptive Iterative Simulated Annealing
ALNS	Adaptive Large Neighbourhood Search
BB	Branch and Bound
BC	Branch and Cut
BCP	Branch and Cut and Price
BP	Branch And Price
CRP	Component Relocation Problem
CVRP	Capacitated Vehicle Routing Problem
ESPPRC	Elementary Shortest Path Problem with Resources Constraints
FSP	Flow Shop Problem
GA	Genetic Algorithm
GRASP	Greedy Randomised Adaptive Search Procedure
ILS	Iterated Local Search
LNS	Large Neighbourhood Search
LP	Linear Programming
LS	Local Search
MILP	Mixed Integer Linear Programming
REF	Resource Extension Function
RHS	Rolling Horizon Subproblem
SA	Simulated Annealing
SEC	Subtour Elimination Constraint
SIRP	Ship Installation Routing Problem
SP	Set Partitioning
TSP	Traveling Salesman Problem
VND	Variable Neighbourhood Descent
VNS	Variable Neighbourhood Search
VRP	Vehicle Routing Problem

VRPTT Vehicle Routing Problem with Trailers and Transshipments

VRPTW Vehicle Routing Problem with Time Windows

WSS Warm Start Subproblem

Variables CRP

B Batches

B' Batches in subproblem

b^a Arrival batches per component

b^d Departure batches per component

C Clusters in direct unloading algorithm

c Clustering variable

c Decision variable for clustering

C^e Enclosed clusters in direct unloading algorithm

C^f Free clusters in direct unloading algorithm

f Blocking variable

f Decision variable for blocking

M^h Locations with front neighbours

M^s Locations with side neighbours

N Components

N'^a Arriving components in subproblem

N'^d Departing components in subproblem

N^a Arriving components

N^B Components per batch

N^d Departing components

N^u Movable components in direct unloading algorithm

N_b^{d+} Components departing after batch b

P Allowed paths

p Penalty for unavailable space

p^c Penalty for clustering

p^u Penalty for unavailable space

P^{bt_b} Paths corresponding to batch type of b

R^h Front neighbouring locations

R^s Side neighbouring locations

R_i All neighbouring locations of location i

S Locations

S_r^c Locations of cluster r

s^d Vessel location

S^s Storage locations

S^u Locations unoccupied in direct unloading algorithm

S^{sl} Loadout locations

S^{st} Storage-only locations

T Timesteps

t^e time after last timestep ($t^e = t^{max} + 1$)

tb_b^{max} latest departure time for batch b

tb_b^{min} Earliest arrival time for batch b

x Decision variable for relocations

y Helper variable for positions

Definitions

Blocked location Location which is inaccessible by crane due to surrounding components

Deterministic Time Measure for computational time

Efficient Taking polynomial (or less) time

Free side Having a free side means having an unoccupied location directly next to it. If this location is at the back row, there must also be an unoccupied location in front of it.

Integral Relaxations Removing the requirement of integrality

Intrafield travelling Travelling between turbines in the installation field

Outerfield travelling Travelling between the harbour and the installation field

Pre-marshalling Preparing storage field for the next loading or unloading action

Relevant side neighbour A side neighbour of location i, which has to be considered in determining if location i is blocked

Self blocking The principle that when location j is blocked, but would not be if there is no component at neighbour i, then relocation from i to j is allowed.

Tour Complete path of vehicle consisting of multiple trips

Trip Path of vehicle between leaving from and returning to the depot without any intermediate stop at a depot

Variables SIRP

c Arc costs

c^i Installation durations

Di Installation precedence sets

Ds Ship precedence set

e Decision variable: ending time of project

es Ship ending times

H	Large number representing infinity
K	Ships / objects
K	Ships per method
K^D	Installation precedence ship sets
k^D	Ship precedence ships
K^i	Installation ships
K^p	Plugs
M	Installation methods
N	Nodes
n^0	Departing harbour node
n^e	Arriving harbour node
p	Decision variable: Installation methods
pc	Project cost
Q	Capacities
S	Method steps
SA'	Outer simulated annealing subalgorithm
SA''	Inner simulated annealing subalgorithm
sc	Ship costs
ss	Ship starting times
T	simulated annealing temperature
t	time
t^d	Ship precedence offset
ti	Node starting time
tr	Trip starting time
wa	Waiting arc variable
wn	Waiting node variable
x	Decision variable: Arc selection
y	Visits to turbines

Bibliography

El Houssaine Aghezzaf, Birger Raa, and Hendrik Van Landeghem. Modeling inventory routing problems in supply chains of high consumption products. *Eur. J. Oper. Res.*, 169(3):1048–1063, 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.02.008.

Reza H Ahmadi. Scheduling in network flow shops. *J. Glob. Optim.*, 9:293–320, 1996. ISSN 09255001. doi: 10.1007/BF00121676. URL http://www.springerlink.com/content/g803v0683j19230p/?p=e5c5a647f471401a814935e477fab515{&}pi=4.

Abderrahim Ait-Alla, Moritz Quandt, and Michael Lütjen. Aggregate installation planning of offshore wind farms. *Int. J. Energy*, 7(2):23–30, 2013. URL http://www.naun.org/main/NAUN/energy/b012001-201.pdf.

Syrine Roufaida Ait Haddadene, Nacima Labadie, and Caroline Prodhon. A GRASP × ILS for the vehicle routing problem with time windows, synchronization and precedence constraints. *Expert Syst. Appl.*, 66:1339–1351, 2016. ISSN 09574174. doi: 10.1016/j.eswa.2016.09.002. URL http://dx.doi.org/10.1016/j.eswa.2016.09.002.

Attahiru Sule Alfa, Sundresh S. Heragu, and Mingyuan Chen. A 3-OPT based simulated annealing algorithm for vehicle routing problems. *Comput. Ind. Eng.*, 21(1-4):635–639, jan 1991. ISSN 03608352. doi: 10.1016/0360-8352(91)90165-3. URL http://www.sciencedirect.com/science/article/pii/0360835291901653http://linkinghub.elsevier.com/retrieve/pii/0360835291901653.

F. Alonso, M. J. Alvarez, and J. E. Beasley. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *J. Oper. Res. Soc.*, 59(7):963–976, 2008. ISSN 01605682. doi: 10.1057/palgrave.jors.2602405.

P Augerat, Jm Belenguer, E Benavent, A Coberan, D Naddef, and G Rinaldi. Computational results with a branch-and-cut code for the capacitated vehicle routing problem. *Rapp. Rech. - IMAG*, 95(949-M):30, 1998. URL http://cat.inist.fr/?aModele=afficheN{&}cpsidt=51607.

Nabila Azi, Michel Gendreau, and Jean Yves Potvin. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *Eur. J. Oper. Res.*, 178(3):755–766, 2007. ISSN 03772217. doi: 10.1016/j.ejor.2006.02.019.

R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation. *Oper. Res.*, 52(5):723–738, 2004. ISSN 0030-364X. doi: 10.1287/opre.1040.0111. URL http://pubsonline.informs.org/doi/abs/10.1287/opre.1040.0111.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *Eur. J. Oper. Res.*, 218(1):1–6, 2012. ISSN 03772217. doi: 10.1016/j.ejor.2011.07.037. URL http://dx.doi.org/10.1016/j.ejor.2011.07.037.

M L Balinski and R E Quandt. On an Integer Program for a Delivery Problem. *Oper. Res.*, 12 (2):300–304, 1964. ISSN 0030-364X. doi: 10.1287/opre.12.2.300.

John E. Bell and Patrick R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Adv. Eng. Informatics*, 18(1):41–48, 2004. ISSN 14740346. doi: 10.1016/j.aei.2004.07.001.

Ulrich Blasum and Winfried Hochstättler. Application of the Branch and Cut Method to the Vehicle Routing Problem. *White Pap.*, pages 1–22, 2002.

Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review, 2016. ISSN 03608352. URL http://dx.doi.org/10.1016/j.cie.2015.12.007.

José Brandão. A tabu search algorithm for the heterogeneous fixed fleet vehicle routing problem. *Comput. Oper. Res.*, 38(1):140–151, 2011. ISSN 03050548. doi: 10.1016/j.cor.2010.04.008.

David Bredström and Mikael Rönnqvist. Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *Eur. J. Oper. Res.*, 191(1):19–29, 2008. ISSN 03772217. doi: 10.1016/j.ejor.2007.07.033.

Katja Buhrkal, Allan Larsen, and Stefan Ropke. The Waste Collection Vehicle Routing Problem with Time Windows in a City Logistics Context. *Procedia - Soc. Behav. Sci.*, 39:241–254, 2012. ISSN 18770428. doi: 10.1016/j.sbspro.2012.03.105. URL http://linkinghub.elsevier.com/retrieve/pii/S1877042812005721.

Héctor J. Carlo, Iris F.A. Vis, and Kees Jan Roodbergen. Storage yard operations in container terminals: Literature overview, trends, and research directions. *Eur. J. Oper. Res.*, 235(2):412–430, jun 2014. ISSN 03772217. doi: 10.1016/j.ejor.2013.10.054. URL http://ac.els-cdn.com/S0377221713008771/1-s2.0-S0377221713008771-main.pdf?{_}tid=8dd68ec2-54c6-11e7-9820-00000aacb35d{&}acdnat=1497859922{_}e14d8412bc92edcca8ba526f89c0e05bhttp://linkinghub.elsevier.com/retrieve/pii/S0377221713008771.

Marco Caserta, Silvia Schwarze, and Stefan Voß. A mathematical formulation and complexity considerations for the blocks relocation problem. *Eur. J. Oper. Res.*, 219(1):96–104, may 2012. ISSN 03772217. doi: 10.1016/j.ejor.2011.12.039. URL http://dx.doi.org/10.1016/j.ejor.2011.12.039http://linkinghub.elsevier.com/retrieve/pii/S0377221711011337.

Diego Cattaruzza, Nabil Absi, Dominique Feillet, and Thibaut Vidal. A memetic algorithm for the Multi Trip Vehicle Routing Problem. *Eur. J. Oper. Res.*, 236(3):833–848, 2014. ISSN 03772217. doi: 10.1016/j.ejor.2013.06.012. URL http://dx.doi.org/10.1016/j.ejor.2013.06.012.

Diego Cattaruzza, Nabil Absi, and Dominique Feillet. Vehicle routing problems with multiple trips. *4or*, 14(3):223–259, 2016. ISSN 16142411. doi: 10.1007/s10288-016-0306-2.

Marielle Christiansen, Kjetil Fagerholt, Bjørn Nygreen, and David Ronen. Ship routing and scheduling in the new millennium. *Eur. J. Oper. Res.*, 228(3):467–483, aug 2013. ISSN 03772217. doi: 10.1016/j.ejor.2012.12.002. URL http://dx.doi.org/10.1016/j.ejor.2012.12.002http://linkinghub.elsevier.com/retrieve/pii/S0377221712009125.

William J Cook, William H Cunningham, William R Pulleyblank, and Alexander Schrijver. Combinatorial optimization. *Comput. Math. with Appl.*, 35(9):139, may 1998. ISSN 08981221. doi: 10.1016/S0898-1221(98)90683-6. URL http://linkinghub.elsevier.com/retrieve/pii/S0898122198906836.

Teodor Gabriel Crainic, Nicoletta Ricciardi, and Giovanni Storchi. Models for Evaluating and Planning City Logistics Systems. *Transp. Sci.*, 43(4):432–454, 2009. ISSN 0041-1655. doi: 10.1287/trsc.1090.0279. URL http://pubsonline.informs.org/doi/abs/10.1287/trsc.1090.0279.

G Dantzig and Delbert Ray Fulkerson. On the max flow min cut theorem of networks. 1955.

G. B. Dantzig and J H Ramser. The Truck Dispatching Problem. *Manage. Sci.*, 6(1):80–91, 1959. ISSN 0025-1909. doi: 10.1287/mnsc.6.1.80. URL http://www.jstor.org/stable/2627477{%}5Cnhttp://pubsonline.informs.org/doi/abs/10.1287/mnsc.6.1.80.

Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Oper. Res.*, 40(2):342–354, apr 1992. ISSN 0030-364X. doi: 10.1287/opre.40.2.342. URL http://pubsonline.informs.org/doi/abs/10.1287/opre.40.2.342.

Jacques Desrosiers. *Fleet Management and Logistics*. Number January. Springer US, Boston, MA, 1998. ISBN 978-1-4613-7637-8. doi: 10.1007/978-1-4615-5755-5. URL http://link.springer.com/10.1007/978-1-4615-5755-5.

E W Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, dec 1959. ISSN 0029-599X. doi: 10.1007/BF01386390. URL http://link.springer.com/10.1007/BF01386390.

M. Drexl. Synchronization in Vehicle Routing–A Survey of VRPs with Multiple Synchronization Constraints. *Transp. Sci.*, 46(3):297–316, 2012. ISSN 0041-1655. doi: 10.1287/trsc.1110.0400.

Michael Drexl. *On Some Generalized Routing Problems*. 2007.

Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *J. Res. Natl. Bur. Stand. Sect. B Math. Math. Phys.*, 69B(1 and 2):125, 1965. ISSN 0022-4340. doi: 10.6028/jres.069B.013.

Jalel Euchi and Habib Chabchoub. A hybrid tabu search to solve the heterogeneous fixed fleet vehicle routing problem. *Logist. Res.*, 2(1):3–11, 2010. ISSN 18650368. doi: 10.1007/s12159-010-0028-3.

Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4or*, 8(4):407–424, 2010. ISSN 16194500. doi: 10.1007/s10288-010-0130-z.

Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. ISSN 00283045. doi: 10.1002/net.20033.

Thomas A. Feo and Mauricio G C Resende. Greedy Randomized Adaptive Search Procedures. *J. Glob. Optim.*, 6(2):109–133, mar 1995. ISSN 0925-5001. doi: 10.1007/BF01096763. URL http://link.springer.com/10.1007/BF01096763.

Martina Fischetti and Michele Monaci. Proximity search heuristics for wind farm optimal layout. *J. Heuristics*, 22(4):459–474, 2016. ISSN 15729397. doi: 10.1007/s10732-015-9283-4.

Martina Fischetti and David Pisinger. Optimizing wind farm cable routing considering power losses. *Eur. J. Oper. Res.*, 0:1–14, 2017. ISSN 03772217. doi: 10.1016/j.ejor.2017.07.061. URL http://dx.doi.org/10.1016/j.ejor.2017.07.061.

Florian Forster and Andreas Bortfeldt. A tree search heuristic for the container retrieval problem. In *Oper. Res. Proc. 2011*, pages 257–262. 2012. doi: 10.1007/978-3-642-29210-1_41. URL http://link.springer.com/10.1007/978-3-642-29210-1{_}41.

Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi De Aragao, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Math. Program.*, 106(3):491–511, 2006. ISSN 00255610. doi: 10.1007/s10107-005-0644-x.

B Gillett and L Miller. A Heuristic algorithm for the vehicle dispatch problem. *Eur. J. Oper. Res.*, 22(May 2018):340–349, 1974. ISSN 0030-364X. doi: 10.1287/opre.22.2.340.

Fred Glover. HEURISTICS FOR INTEGER PROGRAMMING USING SURROGATE CONSTRAINTS. *Decis. Sci.*, 8(1):156–166, jan 1977. ISSN 0011-7315. doi: 10.1111/j.1540-5915.1977.tb01074.x. URL http://doi.wiley.com/10.1111/j.1540-5915.1977.tb01074.x.

Fred Glover. Tabu Search—Part I. *ORSA J. Comput.*, 1(3):190–206, aug 1989. ISSN 0899-1499. doi: 10.1126/science.220.4598.671. URL http://link.springer.com/10.1007/978-1-4419-1153-7http://pubsonline.informs.org/doi/abs/10.1287/ijoc.1.3.190.

Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Comput. Oper. Res.*, 11(1):49–66, jan 1984. ISSN 03050548. doi: 10.1016/0305-0548(84)90007-8. URL http://linkinghub.elsevier.com/retrieve/pii/0305054884900078.

Luis Gouveia and Jose Manuel Pires. The asymmetric travelling salesman problem and a reformulation of the Miller-Tucker-Zemlin constraints. *Eur. J. Oper. Res.*, 112(1):134–146, 1999. ISSN 03772217. doi: 10.1016/S0377-2217(97)00358-5.

Tore Grünert and H. J. Sebastian. Planning models for long-haul operations of postal and express shipment companies. *Eur. J. Oper. Res.*, 122(2):289–309, 2000. ISSN 03772217. doi: 10.1016/S0377-2217(99)00234-9.

Jatinder N D Gupta. FLOWSHOP SCHEDULES WITH SEQUENCE DEPENDENT SETUP TIMES. *J. Oper. Res.*, 29(3):206–219, 1986.

Jorgen Haahr and Richard M. Lusby. Integrating rolling stock scheduling with train unit shunting. *Eur. J. Oper. Res.*, 259(2):452–468, 2017. ISSN 03772217. doi: 10.1016/j.ejor.2016.10.053.

J.T. Haahr, R.M. Lusby, and J.C. Wagenaar. Optimization methods for the Train Unit Shunting Problem. *Eur. J. Oper. Res.*, 262(3):981–995, 2017. ISSN 03772217. doi: 10.1016/j.ejor.2017.03.068.

Jinil Han, Chungmok Lee, and Sungsoo Park. A Robust Scenario Approach for the Vehicle Routing Problem with Uncertain Travel Times. *Transp. Sci.*, 48(3):373–390, 2014. ISSN 0041-1655. doi: 10.1287/trsc.2013.0476. URL http://pubsonline.informs.org/doi/abs/10.1287/trsc.2013.0476.

Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, dec 2008. ISSN 1619-4500. doi: 10.1007/s10288-008-0089-1. URL http://link.springer.com/10.1007/s10479-009-0657-6http://link.springer.com/10.1007/s10288-008-0089-1.

Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.*, 4(2):100–107, 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136. URL http://portal.acm.org/citation.cfm?doid=1056777.1056779http://ieeexplore.ieee.org/document/4082128/.

F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud. A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4or*, 12(3):235–259, 2014. ISSN 16142411. doi: 10.1007/s10288-013-0238-z.

IEA. World Energy Outlook 2017. *Int. Energy Agency*, 2017. ISSN <null>. doi: 10.1016/0301-4215(73)90024-4. URL http://www.iea.org/Textbase/npsum/weo2017SUM.pdf.

Chandra Ade Irawan, Dylan Jones, and Djamila Ouelhadj. Bi-objective optimisation model for installation scheduling in offshore wind farms. *Comput. Oper. Res.*, 78:393–407, feb 2017. ISSN 03050548. doi: 10.1016/j.cor.2015.09.010. URL http://dx.doi.org/10.1016/j.cor.2015.09.010http://linkinghub.elsevier.com/retrieve/pii/S0305054815002312.

Siti Nurbaya Ismail, Ku Ruhana Ku-Mahamud, and Syariza Abdul-Rahman. A review on delivery routing problem and its approaches, 2017. ISSN 18173195.

Bardia Kamrad, Akhtar Siddque, and Ricardo Ernst. Models and Algorithms for Shuffling Problems in Steel Plants. *Nav. Res. Logist.*, 55(April 2007):541–550, 2012. ISSN 0894069X. doi: 10.1002/nav. URL http://www.interscience.wiley.com/jpages/0894-069X/.

Kap Hwan Kim and Gyu Pyo Hong. A heuristic rule for relocating blocks. *Comput. Oper. Res.*, 33(4):940–954, 2006. ISSN 03050548. doi: 10.1016/j.cor.2004.08.005.

Scott Kirkpatrick, C. D. Gelatt, and M P Vecchi. Optimization by Simulated Annealing. *Science (80-.).*, 220(4598):671–680, may 1983. ISSN 0036-8075. doi: 10.1126/science.220.4598.671. URL http://www.sciencemag.org/cgi/doi/10.1126/science.220.4598.671.

Cagri Koc and Ismail Karaoglan. A Branch and Cut Algorithm for the Vehicle Routing Problem With Multiple Use of Vehicles. *Proc. 41st Int. Conf. Comput. Ind. Eng.*, pages 554–559, 2011.

Çağrı Koç, Tolga Bektaş, Ola Jabali, and Gilbert Laporte. Thirty years of heterogeneous vehicle routing. *Eur. J. Oper. Res.*, 249(1):1–21, feb 2016. ISSN 03772217. doi: 10.1016/j.ejor.2015.07.020. URL http://linkinghub.elsevier.com/retrieve/pii/S0377221715006530.

Suresh Nanda Kumar and Ramasamy Panneerselvam. A Survey on the Vehicle Routing Problem and Its Variants. *Intell. Inf. Manag.*, 04(03):66–74, 2012. ISSN 2160-5912. doi: 10.4236/iim.2012.43010. URL http://www.scirp.org/journal/doi.aspx?DOI=10.4236/iim.2012.43010.

Jana Lehnfeld and Sigrid Knust. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *Eur. J. Oper. Res.*, 239(2):297–312, 2014. ISSN 03772217. doi: 10.1016/j.ejor.2014.03.011. URL http://dx.doi.org/10.1016/j.ejor.2014.03.011http://ac.els-cdn.com/S0377221714002252/1-s2.0-S0377221714002252-main.pdf?{_}tid=66ba2c22-54c6-11e7-8dee-00000aab0f01{&}acdnat=1497859856{_}99568dfb40fa5b4a85013ddae89dbc50.

Adam N. Letchford, Richard W. Eglese, and Jens Lysgaard. Multistars, partial multistars and the capacitated vehicle routing problem. *Math. Program. Ser. B*, 94(1):21–40, 2002. ISSN 00255610. doi: 10.1007/s10107-002-0336-8.

Helena R. Lourenço, Olivier C Martin, and Thomas Stützle. Iterated Local Search: Framework and Applications. volume 146, pages 363–397. 2010. ISBN 978-1-4419-1663-1. doi: 10.1007/978-1-4419-1665-5_12. URL http://link.springer.com/10.1007/978-1-4419-1665-5http://link.springer.com/10.1007/978-1-4419-1665-5{_}12.

Chao Lu, Ruiyou Zhang, and Shixin Liu. A 0-1 integer programming model and solving strategies for the slab storage problem. *Int. J. Prod. Res.*, 54(8):2366–2376, 2016. ISSN 0020-7543. doi: 10.1080/00207543.2015.1076949. URL http://www.tandfonline.com/doi/full/10.1080/00207543.2015.1076949.

Quan Lu and Maged M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *Eur. J. Oper. Res.*, 175(2):672–687, 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.05.012.

Michael Lütjen and Hamid Reza Karimi. Approach of a Port Inventory Control System for the Offshore Installation of Wind Turbines. *Isope*, 4(January 2012):502–508, 2012. ISSN 10986189.

Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Program.*, 100(2):423–445, jun 2004. ISSN 0025-5610. doi: 10.1007/s10107-003-0481-8. URL http://link.springer.com/10.1007/s10107-003-0481-8.

Federico Malucelli, Stefano Pallottino, and Daniele Pretolani. The stack loading and unloading problem. *Discret. Appl. Math.*, 156(17):3248–3266, 2008. ISSN 0166218X. doi: 10.1016/j.dam.2008.05.020. URL http://dx.doi.org/10.1016/j.dam.2008.05.020.

Sanjay V. Mehta. Predictable scheduling of a single machine subject to breakdowns. *Int. J. Comput. Integr. Manuf.*, 12(1):15–38, jan 1999. ISSN 0951-192X. doi: 10.1080/095119299130443. URL http://www.tandfonline.com/doi/abs/10.1080/095119299130443.

C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, 7(4):326–329, 1960. ISSN 00045411. doi: 10.1145/321043.321046. URL http://portal.acm.org/citation.cfm?doid=321043.321046.

N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997. ISSN 03050548. doi: 10.1016/S0305-0548(97)00031-2. URL http://linkinghub.elsevier.com/retrieve/pii/S0305054897000312.

Zahra Naji-Azimi and Majid Salari. A complementary tool to enhance the effectiveness of existing methods for heterogeneous fixed fleet vehicle routing problem. *Appl. Math. Model.*, 37(6):4316–4324, 2013. ISSN 0307904X. doi: 10.1016/j.apm.2012.09.027. URL http://dx.doi.org/10.1016/j.apm.2012.09.027.

Jungsup Oh, Jongmoon Baik, and Sung Hwa Lim. A model independent S/W framework for search-based software testing. *Sci. World J.*, 2014(September), 2014. ISSN 1537744X. doi: 10.1155/2014/126348.

Manfred Padberg and Giovanni Rinaldi. A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Rev.*, 33(1):60–100, mar 1991. ISSN 0036-1445. doi: 10.1137/1033004. URL http://epubs.siam.org/doi/10.1137/1033004.

G Abor Pataki. The bad and the good-and-ugly: formulations for the traveling salesman problem. *Citeseer*, pages 1–6, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7256{&}rep=rep1{&}type=pdf.

R. J. Petch and S. Salhi. A multi-phase constructive heuristic for the vehicle routing problem with multiple trips. *Discret. Appl. Math.*, 133(1-3):69–92, 2003. ISSN 0166218X. doi: 10.1016/S0166-218X(03)00434-7.

Matthew E H Petering and Mazen I. Hussein. A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem. *Eur. J. Oper. Res.*, 231(1):120–130, 2013. ISSN 03772217. doi: 10.1016/j.ejor.2013.05.037. URL http://dx.doi.org/10.1016/j.ejor.2013.05.037.

David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Comput. Oper. Res.*, 34(8):2403–2435, 2007. ISSN 03050548. doi: 10.1016/j.cor.2005.09.012.

Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Comput. Oper. Res.*, 31(12):1985–2002, oct 2004. ISSN 03050548. doi: 10.1016/S0305-0548(03)00158-8. URL http://linkinghub.elsevier.com/retrieve/pii/S0305054803001588.

R Z Rios-Mercado and Jonathan F Bard. The flow shop scheduling polyhedron with setup times. *J. Comb. Optim.*, 7(3):291–318, 2003.

Roger Z. Ríos-Mercado and Jonathan F Bard. An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Times. *J. Heuristics*, 5(1):53–70, 1999. ISSN 13811231. doi: 10.1023/A:1009691028143. URL http://link.springer.com/10.1023/A:1009691028143.

ROGER Z. RÍOS-MERCADO and JONATHAN F. BARD. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. *IIE Trans.*, 31(8):721–731, 1999. ISSN 0740817X. doi: 10.1023/A:1007650011043. URL http://link.springer.com/10.1023/A:1007650011043.

Juan Carlos Rivera, H. Murat Afsar, and Christian Prins. A multistart iterated local search for the multitrip cumulative capacitated vehicle routing problem. *Comput. Optim. Appl.*, 61(1):159–187, may 2015. ISSN 0926-6003. doi: 10.1007/s10589-014-9713-5. URL http://link.springer.com/10.1007/s10589-014-9713-5.

Stefan Ropke. Heuristic and exact algorithms for vehicle routing problems. *Unpubl. PhD thesis, Comput. Sci. Dep. Univ. Copenhagen*, (December):256, 2005. URL http://www.diku.dk/{~}sropke/Papers/PHDThesis.pdf.

Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transp. Sci.*, 40(4):455–472, 2006. ISSN 0041-1655. doi: 10.1287/trsc.1050.0135. URL http://pubsonline.informs.org/doi/abs/10.1287/trsc.1050.0135.

Stuart A. Mitchell Roy, J.S. PuLP, 2017. URL https://github.com/coin-or/pulp.

M. Angélica Salazar-Aguilar, André Langevin, and Gilbert Laporte. The synchronized arc and node routing problem: Application to road marking. *Comput. Oper. Res.*, 40(7):1708–1715, 2013. ISSN 03050548. doi: 10.1016/j.cor.2013.01.007.

S. Salhi and R. J. Petch. A GA based heuristic for the vehicle routing problem with multiple trips. *J. Math. Model. Algorithms*, 6(4):591–613, 2007. ISSN 15701166. doi: 10.1007/s10852-007-9069-2.

Bernd Scholz-Reiter, Jens Heger, Michael Lütjen, and Anne Schweizer. A milp for installation scheduling of offshore wind farms. *Int. J. Math. Model. Methods Appl. Sci.*, 5(2):371–378, 2011. ISSN 19980140.

Kenneth Sörensen and Fred W. Glover. Metaheuristics. In Saul I. Gass and Michael C. Fu, editors, *Encycl. Oper. Res. Manag. Sci.*, number April 2016, pages 960–970. Springer US, Boston, MA, 2013. ISBN 978-1-4419-1137-7. doi: 10.1007/978-1-4419-1153-7_1167. URL http://link.springer.com/10.1007/978-1-4419-1153-7http://link.springer.com/10.1007/978-1-4419-1153-7{_}1167.

Magnus Stålhane, Lars Magnus Hvattum, and Vidar Skaar. Optimization of Routing and Scheduling of Vessels to Perform Maintenance at Offshore Wind Farms. *Energy Procedia*, 80(1876):92–99, 2015. ISSN 18766102. doi: 10.1016/j.egypro.2015.11.411. URL http://linkinghub.elsevier.com/retrieve/pii/S1876610215021438.

E Taillard. Parallel Iterative Search Methods for Vehicle-Routing Problems. *Networks*, 23(8):661–673, 1993. doi: DOI10.1002/net.3230230804.

É. D. Taillard, G. Laporte, and M. Gendreau. Vehicle routing with multiple use of vehicles. *J. Oper. Res. Soc.*, 47(March 1995):1065–1070, 1996. ISSN 0160-5682. URL http://www.palgrave-journals.com/jors/journal/v47/n8/abs/jors1996133a.html.

Lixin Tang and Lin Huang. Optimal and near-optimal algorithms to rolling batch scheduling for seamless steel tube production. *Int. J. Prod. Econ.*, 105(2):357–371, feb 2007. ISSN 09255273. doi: 10.1016/j.ijpe.2004.04.011. URL http://linkinghub.elsevier.com/retrieve/pii/S0925527305002343.

C. D. Tarantilis, C. T. Kiranoudis, and V. S. Vassiliadis. A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. *Eur. J. Oper. Res.*, 152(1):148–158, 2004. ISSN 03772217. doi: 10.1016/S0377-2217(02)00669-0.

Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem, 2002. ISSN 0166218X.

Pamela H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Comput. Optim. Appl.*, 9(3):211–228, 1998. ISSN 09266003. doi: 10.1023/A:1018346107246.

Iris F.A. Vis and Reneé De Koster. Transshipment of containers at a container terminal: An overview. *Eur. J. Oper. Res.*, 147(1):1–16, 2003. ISSN 03772217. doi: 10.1016/S0377-2217(02)00293-X.

Yat-wah Wan, Jiyin Liu, and Pei-Chun Tsai. The assignment of storage locations to containers for a container stack. *Nav. Res. Logist.*, 56(8):699–713, dec 2009. ISSN 0894069X. doi: 10.1002/nav.20373. URL http://www.interscience.wiley.com/jpages/0894-069X/http://doi.wiley.com/10.1002/nav.20373.

Thomas Winter and Uwe T Zimmermann. Real-time dispatch of trams in storage yards *. *Ann. Oper. Res.*, 96:287–315, 2000. ISSN 02545330. doi: 10.1023/A:1018907720194.

Hande Yaman. Formulations and valid inequalities for the heterogeneous vehicle routing problem. *Math. Program.*, 106(2):365–390, 2006. ISSN 00255610. doi: 10.1007/s10107-005-0611-6.

Yujun Yang, Ye Chen, and Chuanze Long. Flexible robotic manufacturing cell scheduling problem with multiple robots. *Int. J. Prod. Res.*, 7543(July):0, 2017. ISSN 0020-7543. doi: 10.1080/00207543.2016.1176267. URL http://dx.doi.org/10.1080/00207543.2016.1176267.

Victor Yaurima, Larisa Burtseva, and Andrei Tchernykh. Computers & Industrial Engineering Hybrid flowshop with unrelated machines , sequence-dependent setup time , availability constraints and limited buffers q. *Comput. Ind. Eng.*, 56(4):1452–1463, 2009. ISSN 0360-8352. doi: 10.1016/j.cie.2008.09.004. URL http://dx.doi.org/10.1016/j.cie.2008.09.004.

Miao Yu, Viswanath Nagarajan, and Siqian Shen. Minimum makespan vehicle routing problem with compatibility constraints. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 10335 LNCS, pages 244–253, 2017. ISBN 9783319597751. doi: 10.1007/978-3-319-59776-8_20.

Wei Yu, Zhaohui Liu, Leiyang Wang, and Tijun Fan. Routing open shop and flow shop scheduling problems. *Eur. J. Oper. Res.*, 213(1):24–36, 2011. ISSN 03772217. doi: 10.1016/j.ejor.2011.02.028. URL http://dx.doi.org/10.1016/j.ejor.2011.02.028.

Zhen You Zhang. Scheduling and Routing Optimization of Maintenance Fleet for Offshore Wind Farms Using Duo-ACO. *Adv. Mater. Res.*, 1039:294–301, oct 2014. ISSN 1662-8985. doi: 10.4028/www.scientific.net/AMR.1039.294. URL http://www.scientific.net/AMR.1039.294.

Appendices

A

Analysis Walney project

This appendix contains detailed information on the original planning and on the execution of ship routing and installation schedule of the Walney project. The routes, along with travel durations, are given in Appendix A.1. Furthermore, in Appendix A.2 and Appendix A.3 the waiting times and costs are given, respectively.

A.1. Routes

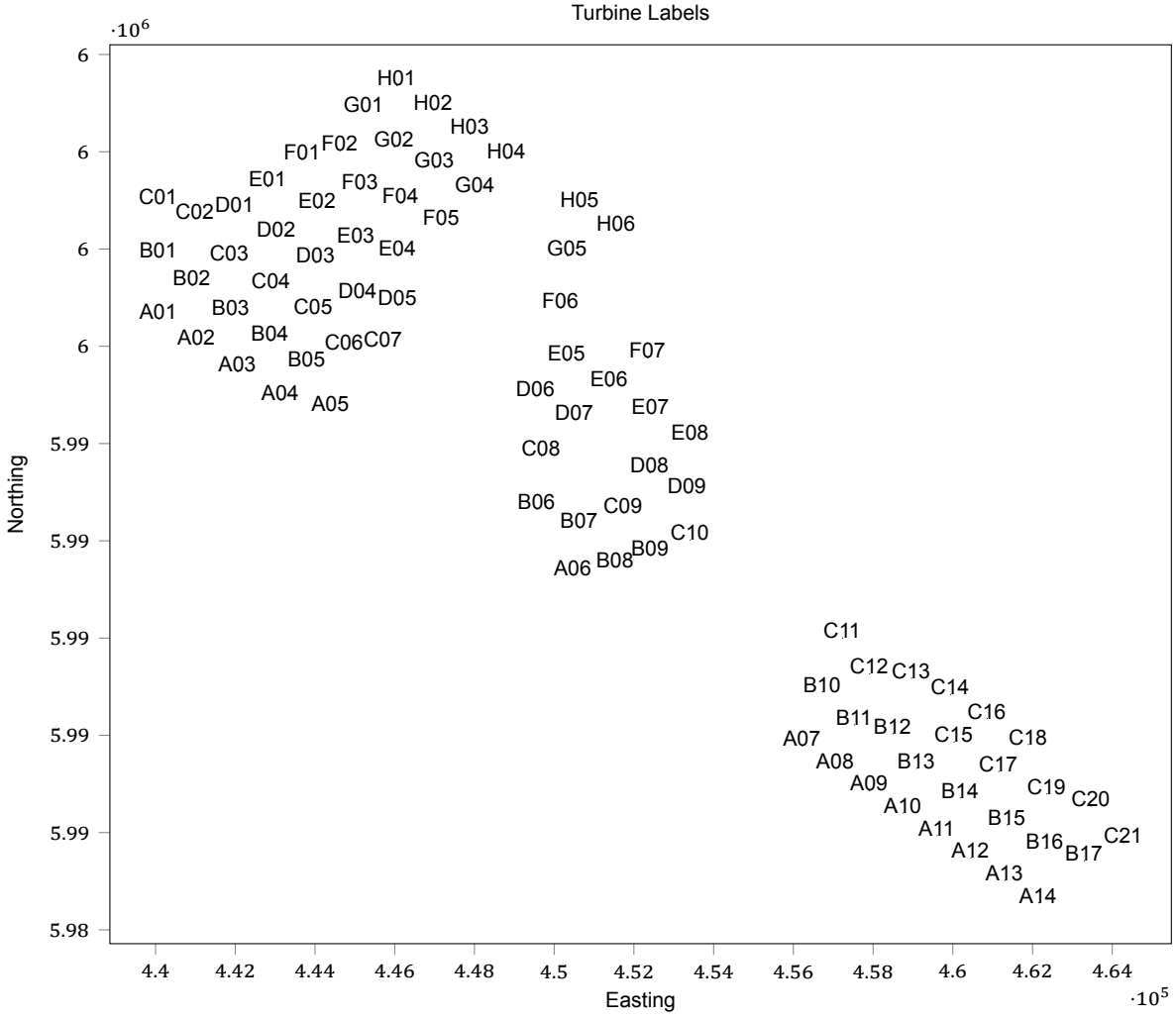


Figure A.1: Labels of Walney Installation field

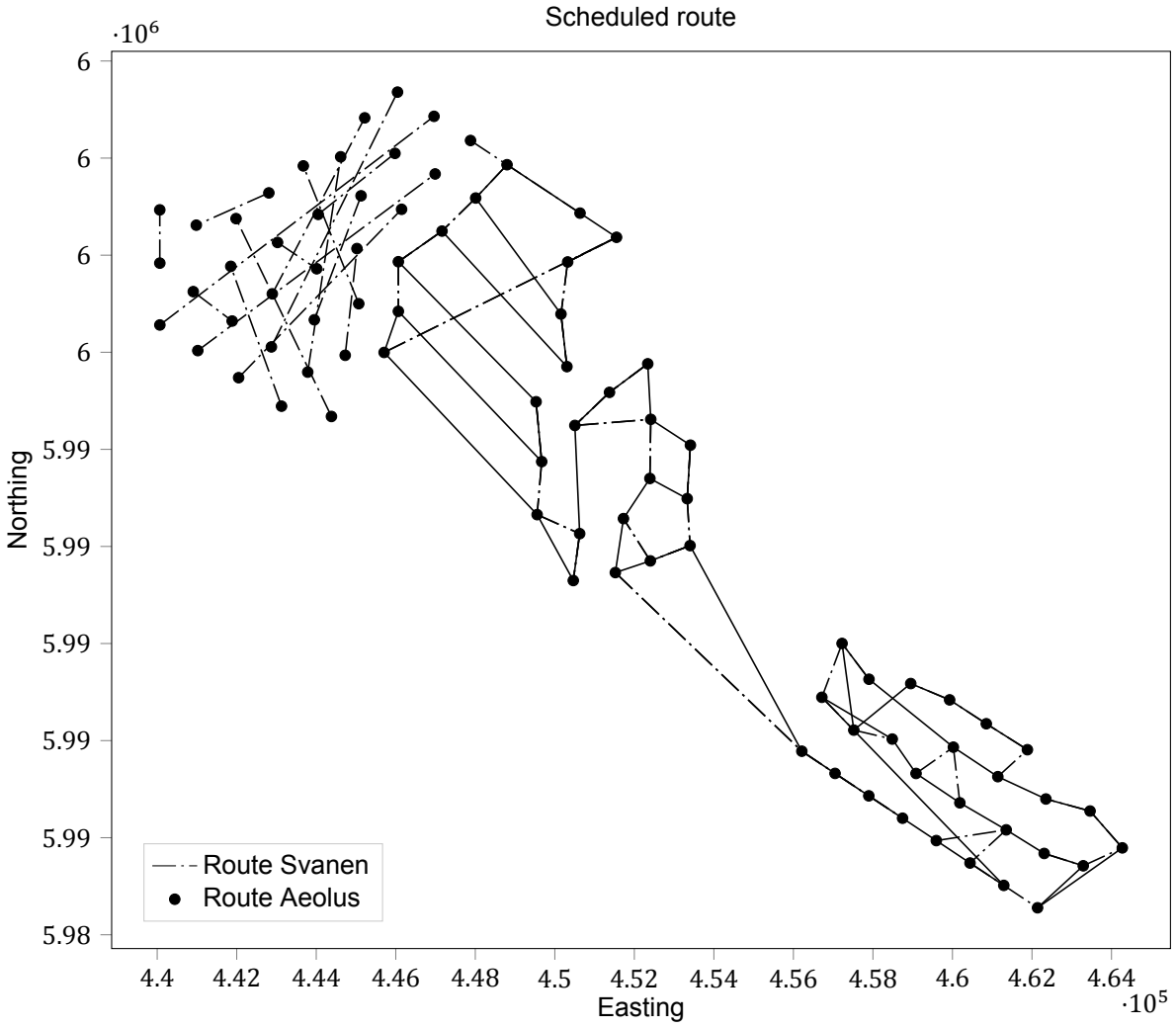


Figure A.2: Scheduled Walney route

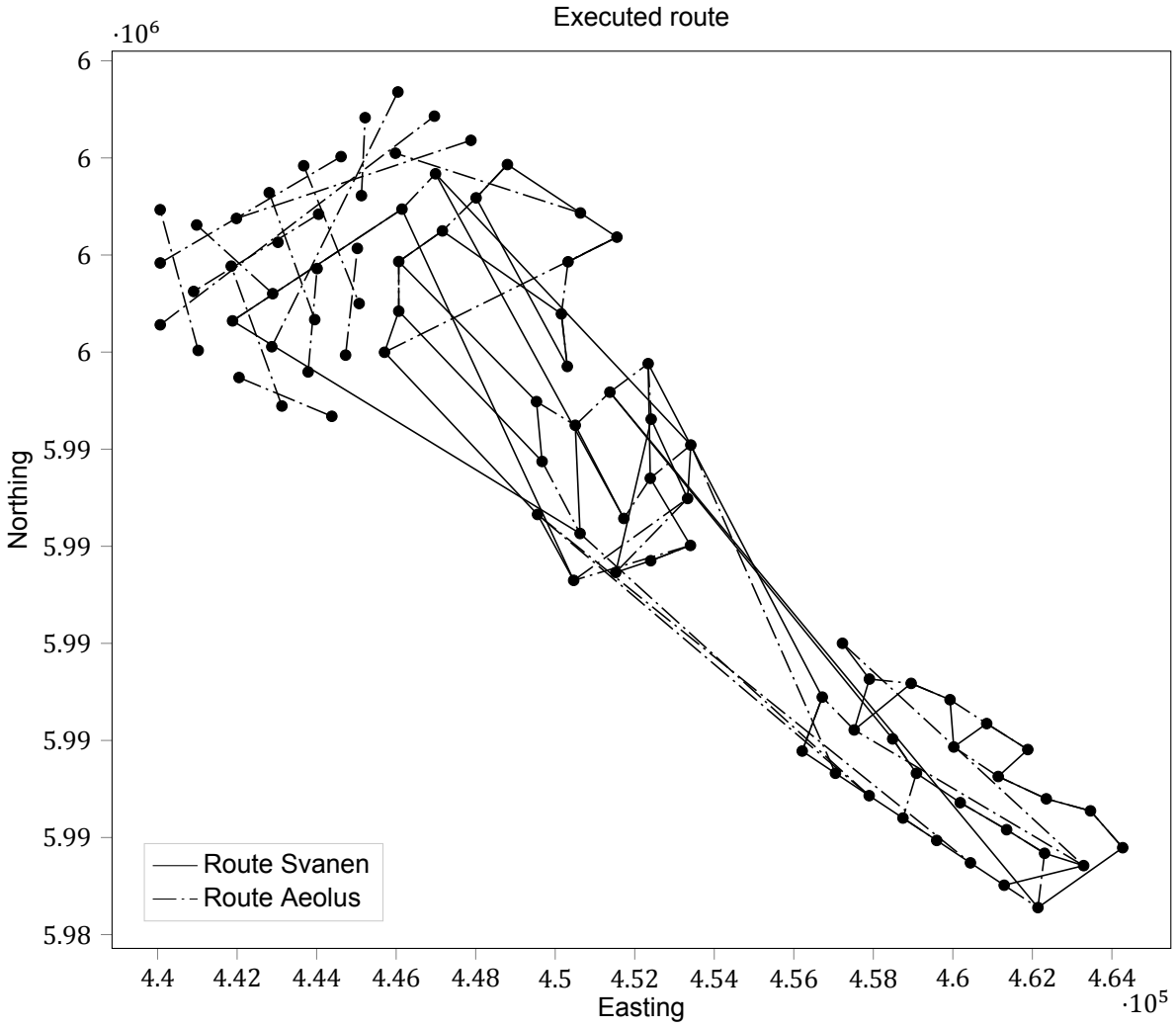


Figure A.3: Executed Walney route

A.1.1. Aeolus travel durations

from	to	duration (min)
C06	E03	55
C05	E01	10
B05	D03	25
B01	F02	85
C02	C04	80
F01	D04	55
A04	C03	75
E02	B02	40
F03	G01	10
B04	H01	45
H05	G02	55
D01	H03	95
A03	A05	80
A02	C01	15
A01	H02	90
B03	F04	57
F04	G03	35
H04	G04	40
G04	F05	95
F05	E04	15
E04	D05	35
D05	E04	25
C07	H06	115
H06	G05	25
G05	F06	25
F06	E05	50
A08	E08	95
E08	D08	45
D08	C09	40
D06	D07	80
D07	E06	30
E06	F07	60
F07	E07	20
B09	C10	25
C10	A06	25
A06	D09	40
D09	B08	40
B12	B13	30
B13	A10	25
A10	A11	50
C08	B07	45
B07	A09	60
A09	B06	85
B06	A12	75
A13	A14	75
A14	B16	60
B16	B15	65
B15	B14	60
C21	C20	35
C20	C19	40
C19	C17	40
C17	C15	30
C11	B17	70
B17	B11	85
B11	B10	25
B10	A07	30
C12	C13	45
C13	C14	40
C14	C16	30
C16	C18	40

Table A.1: Duration of Aeolus intrafield travelling

A.1.2. Svanen travel durations

from	to	duration (min)
G05	H06	55
H06	H04	890
H04	G04	135
G04	E05	40
E05	F06	30
F06	F05	50
F05	E04	20
E04	D06	60
D06	C08	20
C08	D05	35
D05	C07	60
C07	B06	95
B06	A06	90
A06	F04	70
F04	B03	180
B03	B07	110
B07	D07	115
D07	C09	70
C09	G03	75
G03	E08	70
E08	D09	25
D09	E07	30
E07	B08	450
B08	B09	20
B09	C10	17
C10	D08	30
D08	F07	40

F07	B10	62
B10	A07	20
A07	A08	20
A08	A09	28
A09	A10	20
A10	A11	25
A11	A12	19
A12	A13	4
A13	B17	380
B17	B16	20
B16	B15	19
B15	B14	13
B14	B13	5
B13	B12	25
B12	E06	115
E06	A14	140
A14	C21	65
C21	C20	95
C20	C19	40
C19	C17	21
C17	C18	13
C18	C16	5
C16	C15	16
C15	C14	25
C14	C13	15
C13	B11	25
B11	C12	95
C12	C11	20

Table A.2: Executed intrafield route of Svanen

A.2. Waiting

Turbine	Time (min)
G05	2440
H04	120
F05	150
C07	60
A06	35
B03	15
B07	27
D07	45
C09	80
E08	45
D09	225
A07	95
A09	480
A10	325
A11	195
A12	690
B17	25
B15	25
B13	425
C11	286
Total	5788

Table A.3: Svanen waiting time due to plugs

A.3. Costs

	Costs (€)
Aeolus per day	150.000
Svanen per day	75.000
Tugboat per day	5.000
Harbour per day	50.000
Crane relocation	1.000

Table A.4: Estimated costs during Walney project

	Intrafield sailing (hr)	Cost (10^3 €)
Svanen intrafield Sailing	71	220
Aeolus intrafield sailing	50	310
Total sailing	120	530
Svanen waiting on plugs	96	301

Table A.5: Costs of intrafield sailing

B

Offshore scheduling by Flowshop approach

The offshore scheduling problem can be seen as a variation of a flow shop problem, or as a variation of a vehicle routing problem. Both approaches were researched based on available literature and it was found that the flow shop approach would require large extensions which could not be found in literature. Therefore, the VRP is considered the best option. The flow shop approach is still discussed partially here since it has potential to solve the installation method decision problem. The first extension to be considered when looking at this problem from a flow shop point of view is the routing flow shop. This is a flow shop where machines or jobs travel between locations.

A generalised version for the routing open shop problem was given by Yu et al. [2011]. An open shop problem is a flow shop without constraints on the order of processes. This means that all processes will have to be finished, but it can be in any order. They also handle a flow shop for just 2 machines and a tree-graph. A tree graph is a graph is a graph where any two vertices are connected by exactly one path, as seen in Figure B.1. For this problem they prove NP-Hardness.

Ahmadi [1996] introduce the network flow shop. This is a flow shop where the jobs travel between machines, instead of the machines between location. The problem might contain some useful elements for the routing flow shop. The paper of Ahmadi focuses mainly on the computational hardness of the problem and proposes a heuristic. The problem where jobs travel between machines is well researched since it is used for manufacturing systems where items are moved by automated guided vehicles.

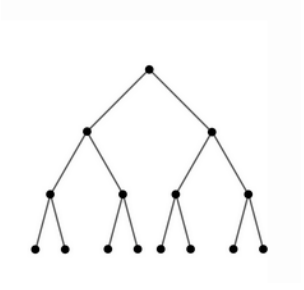


Figure B.1: A tree graph

The flowshop with traveling machines can be seen as a flow-shop with sequence dependent setup times. This problem was introduced by Gupta [1986]. They included setup times for machines based on the job order at that machine. This means that if job j is scheduled after i on machine time m, there will be a setup time of $a_m(i,j)$. This approach can be used to model travelling time for ships. They propose a mathematical model and prove its NP-completeness.

The installation process has similarities with a flow shop problem with sequence dependent durations. Here the duration of certain tasks are depending on the sequence in which the operations are done. This can be used to model the travelling times between locations.

RÍOS-MERCADO and BARD [1999] presented a branch-and-bound algorithm to solve the sequence dependent setup time flow shop scheduling problem. This algorithm included a

custom lower and upper bound and an elimination criterion. In the same year, Bard presented a heuristic procedure for the same problem Ríos-Mercado and Bard [1999]. Later Rios-Mercado and Bard [2003] presented two MILP models for the same problem which was solved using a branch-and-cut model.

Another branch and bound algorithm was presented by Tang and Huang [2007] who studied production scheduling for steel tube production. Next to this they also presented a two-stage heuristic based on a neighbourhood search method.

A method to model the transporting capacity on ships would be by using a constrained buffer. A constrained buffer in a flowshop means that there is a limited amount of jobs waiting between machines. If loading and installing components are seen as machines, then a buffer would model the carrying capacity of transporting ships. Yaurima et al. [2009] used a genetic algorithm to solve the buffer-constrained flowshop problem with sequence dependent setup times.

An important part of the flow shop approach is that machines need to be able to handle multiple tasks. A ship needs to load and install. Travelling can be modelled as a setup time and does not have to be a seperate task. There are studies about job shops for multi-purpose machines (see Yang et al. [2017]), but nothing for flow shops. This is a major gap in required research. It is of course possible that there is some research and it has not been found, but the scarcity suggest that the flow shop approach probably is not the best way to solve the problem.

C

Relocation constraints Monopiles

Similar to the relocation of transition pieces is the relocation of monopiles, also present in the Walney Installation project. These are stored in multiple rows, of which one is shown in Figure C.1. To solve the crane movement problem for monopiles, the same model as introduced in Chapter 6 can be used, but with differen relocation constraints.

In a given row, monopiles can only be placed on the outsides of that row, and only the outer ones can be removed, as shown in Figure C.1. While in theory it is possible to leave empty spaces in a row as shown as in the top row, this will still only allow the outer ones to be accessible and therefore any solution with space in between the rows can be converted to a solution of the same amount of moves by not allowing empty spaces in rows.

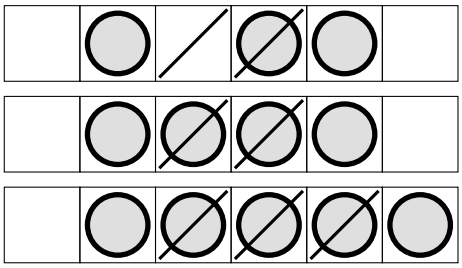


Figure C.1: Rows of monopiles. Diagonal lines show blocked positions

Consider a field with multiple rows $r \in R$ where each row R_r consists of multiple locations $\{sr_1^r, ..., sr_{max}^r\}$ with sr_1^r denoting the leftmost location and sr_{max}^r denoting the rightmost location. All rows combined form the storage field, thus $S^s = \cup_{r \in R} R_r$. The rest of the notation is the same as in Chapter 4.

As stated earlier, MPs can only be relocated from the end of rows to ends of other rows, or to empty rows. Equation (C.1a) blocks relocations from i, if there is at least one other component on any location left of i, and no component directly left of i. Note that the two leftmost locations are not considered in Equation (C.1a) since the leftmost location is never blocked, and the second location always has an adjacent occupied left-neighbour, or no left-neighbours at all.

The converse is done in Equation (C.1b) for components on the right side. Adding these two equations thus define that if there is an adjacent unoccupied location at either side of location i, that complete side must be empty for relocations from i to be non-blocked. Therefore, any component with an adjacent unoccupied location is now blocked unless it is at the first,last or only component in a row.

This leaves components with no adjacent unoccupied locations, and since these components always will be blocked blocks any relocations to i if i has two adjacent occupied neighbours. Since the self-blocking principle does not apply in this problem, blocking constraints for relocating from a locations can simply be added in the same form, resulting in

Equations (C.1d) to (C.1f)

$$\sum_{j\in S}\sum_{n\in N} x_{tnij} \leq 1 - \frac{1}{i-2}\sum_{j=1}^{i-2} y_t j + y_{t(j-1)} \forall r \in R, t \in T, 3 \leq i \leq |R_r| \tag{C.1a}$$

$$\sum_{j\in S}\sum_{n\in N} x_{tnij} \leq 1 - \frac{1}{|R_r|-i-2}\sum_{j=i+2}^{|R_r|} y_{tj} + y_{t(j-1)} \forall r \in R, t \in T, 1 \leq i \leq |R_r| - 2 \tag{C.1b}$$

$$\sum_{j\in S}\sum_{n\in N} x_{tnij} \leq 2 - y_{t(i-1)} - y_{t(i+1)} \forall r \in R, 1 \leq i \leq |R_r| - 1 \tag{C.1c}$$

$$\sum_{j\in S}\sum_{n\in N} x_{tnji} \leq 1 - \frac{1}{i-2}\sum_{j=1}^{i-2} y_{tj} + y_{t(j-1)} \forall r \in R, t \in T, 3 \leq i \leq |R_r| \tag{C.1d}$$

$$\sum_{j\in S}\sum_{n\in N} x_{tnji} \leq 1 - \frac{1}{|R_r|-i-2}\sum_{j=i+2}^{|R_r|} y_{tj} + y_{t(j-1)} \forall r \in R, t \in T, 1 \leq i \leq |R_r| - 2 \tag{C.1e}$$

$$\sum_{j\in S}\sum_{n\in N} x_{tnji} \leq 2 - y_{t(i-1)} - y_{t(i+1)} \forall r \in R, 1 \leq i \leq |R_r| - 1 \tag{C.1f}$$

With the rest of the model as introduced by in Chapter 6 the relocation problem for the monopiles is now defined.

D

Test results

D.1. Crane tests

In this section the results for the CRP are presented. Tests are performed for three types of problems: *loading*, *storage* and *buffer*. The first two types both consider components arriving and departing in batches of 2. The field for these problems consists of 2 loadout locations, and a storage of consisting with the dimensions $(\lceil |N|/2 \rceil, 2)$ (Width, Height). The *buffer* type problems have custom fields and flow schedules per instance, and are given in Table D.1 .

D.1.1. Exact crane optimisation tests

Components	Loadout	Storage-only	Flow schedule
6	2	5	(2 in, 2 in, 2 out, 2 in, 2 out, 2 out)
8	4	5	(4 in, 2 out, 4 in, 2 out, 2 out, 2 out)
12	3	12	(3 in, 3 in, 2 out, 3 in, 3 in, 2 out, 2 out, 2 out, 2 out, 2 out)
18	3	12	(3 in, 3 in, 2 out, 2 in, 3 in, 2 out, 2 out, 3 in, 3 in, 2 out, 2 out, 2 out, 2 out, 2 out, 2 out)

Table D.1: Combined tests

Type	Cuts	Moves	Det. time	Real-time	|t|	Components	Nodes	Start type
Buffer	None	20	37509	394	28	8	317	Warm-start
Buffer	None	20	43112	1210	20	8	2459	Tight-upperbound
Buffer	None	20	46288	602	28	8	770	Standard
Buffer	All	20	145624	3448	28	8	2022	Warm-start
Buffer	All	20	6487	17	20	8	0	Tight-upperbound
Buffer	All	20	112672	1940	28	8	1760	Standard
Buffer	Time-symmetry	20	68637	395	28	8	353	Warm-start
Buffer	Time-symmetry	20	25630	194	20	8	132	Tight-upperbound
Buffer	Time-symmetry	20	126654	3110	28	8	2433	Standard
Buffer	Min-moves	20	51044	429	28	8	537	Warm-start
Buffer	Min-moves	20	25893	533	20	8	953	Tight-upperbound
Buffer	Min-moves	20	160505	5134	28	8	10712	Standard
Buffer	Double-relocate	20	45928	846	28	8	202	Warm-start
Buffer	Double-relocate	20	25153	418	20	8	455	Tight-upperbound
Buffer	Double-relocate	20	67881	1064	28	8	799	Standard
Buffer	None	20	45295	136	29	8	45	Warm-start
Buffer	None	20	4816	15	20	8	0	Tight-upperbound
Buffer	None	20	63272	155	29	8	17	Standard
Buffer	All	20	132484	2699	29	8	2977	Warm-start
Buffer	All	20	28135	441	20	8	669	Tight-upperbound
Buffer	All	20	108696	1378	29	8	1083	Standard
Buffer	Time-symmetry	20	95061	1335	29	8	911	Warm-start
Buffer	Time-symmetry	20	19709	162	20	8	154	Tight-upperbound
Buffer	Time-symmetry	20	114800	1453	29	8	1304	Standard
Buffer	Min-moves	20	93631	1392	29	8	1203	Warm-start
Buffer	Min-moves	20	9332	26	20	8	0	Tight-upperbound
Buffer	Min-moves	20	22890	62	29	8	0	Standard
Buffer	Double-relocate	20	50574	177	29	8	43	Warm-start
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Cuts	Moves	Det. time	Real-time	|t|	Components	Nodes	Start type
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Buffer	Double-relocate	20	30193	496	20	8	873	Tight-upperbound
Buffer	Double-relocate	20	51154	314	29	8	243	Standard
Loading	None	15	2154	2	15	7	0	Warm-start
Loading	None	15	3577	13	15	7	0	Tight-upperbound
Loading	None	15	3577	14	15	7	0	Standard
Loading	All	15	2504	2	15	7	0	Warm-start
Loading	All	15	4552	13	15	7	0	Tight-upperbound
Loading	All	15	4552	14	15	7	0	Standard
Loading	Time-symmetry	15	1733	1	15	7	0	Warm-start
Loading	Time-symmetry	15	2630	7	15	7	0	Tight-upperbound
Loading	Time-symmetry	15	2630	8	15	7	0	Standard
Loading	Min-moves	15	2744	3	15	7	0	Warm-start
Loading	Min-moves	15	3027	7	15	7	0	Tight-upperbound
Loading	Min-moves	15	3027	7	15	7	0	Standard
Loading	Double-relocate	15	4167	10	15	7	4	Warm-start
Loading	Double-relocate	15	11388	42	15	7	89	Tight-upperbound
Loading	Double-relocate	15	11388	44	15	7	89	Standard
Loading	None	22	1205244	46159	22	9	103536	Warm-start
Loading	None	22	680153	23833	22	9	50578	Tight-upperbound
Loading	None	22	680153	23913	22	9	50578	Standard
Loading	All	22	813790	33168	22	9	63713	Warm-start
Loading	All	22	547949	22807	22	9	30448	Tight-upperbound
Loading	All	22	547949	23128	22	9	30448	Standard
Loading	Time-symmetry	22	2389821	112733	22	9	89105	Warm-start
Loading	Time-symmetry	22	1405738	57119	22	9	95372	Tight-upperbound
Loading	Time-symmetry	22	1405738	57169	22	9	95372	Standard
Loading	Min-moves	22	597634	18615	22	9	27560	Warm-start
Loading	Min-moves	22	1358773	69869	22	9	167341	Tight-upperbound
Loading	Min-moves	22	1358773	69457	22	9	167341	Standard
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Cuts	Moves	Det. time	Real-time	|t|	Components	Nodes	Start type
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Loading	Double-relocate	22	2656808	138081	22	9	377260	Warm-start
Loading	Double-relocate	22	2685787	142343	22	9	404817	Tight-upperbound
Loading	Double-relocate	22	2685787	142491	22	9	404817	Standard
Storage	None	26	1827	2	26	7	0	Warm-start
Storage	None	26	114759	2702	26	7	3782	Tight-upperbound
Storage	None	26	114759	2791	26	7	3782	Standard
Storage	All	26	2354	2	26	7	0	Warm-start
Storage	All	26	81743	1704	26	7	1420	Tight-upperbound
Storage	All	26	81743	1645	26	7	1420	Standard
Storage	Time-symmetry	26	2090	2	26	7	0	Warm-start
Storage	Time-symmetry	26	146555	2980	26	7	2895	Tight-upperbound
Storage	Time-symmetry	26	146555	3059	26	7	2895	Standard
Storage	Min-moves	26	1862	2	26	7	0	Warm-start
Storage	Min-moves	26	148117	3623	26	7	3348	Tight-upperbound
Storage	Min-moves	26	148117	3630	26	7	3348	Standard
Storage	Double-relocate	26	2119	2	26	7	0	Warm-start
Storage	Double-relocate	26	555657	19409	26	7	49773	Tight-upperbound
Storage	Double-relocate	26	555657	19903	26	7	49773	Standard
Storage	None	35	3978354	151907	37	9	32718	Warm-start
Storage	None	0	3025304	169796	35	9	46033	Tight-upperbound
Storage	None	35	3789153	161110	37	9	20458	Standard
Storage	All	35	4445783	147973	37	9	21307	Warm-start
Storage	All	0	3131366	167189	35	9	55018	Tight-upperbound
Storage	All	35	4248034	157362	37	9	16651	Standard
Storage	Time-symmetry	36	3943521	161704	37	9	20191	Warm-start
Storage	Time-symmetry	34	2822131	70654	36	9	11634	Tight-upperbound
Storage	Time-symmetry	35	4105496	159457	37	9	19253	Standard
Storage	Min-moves	34	2485009	67206	37	9	19284	Warm-start
Storage	Min-moves	0	3228354	177109	34	9	34023	Tight-upperbound
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Cuts	Moves	Det. time	Real-time	|t|	Components	Nodes	Start type
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Storage	Min-moves	0	3253450	166386	37	9	49834	Standard
Storage	Double-relocate	34	2738373	75617	37	9	13837	Warm-start
Storage	Double-relocate	0	3240360	169590	34	9	59701	Tight-upperbound
Storage	Double-relocate	34	1687102	43346	37	9	5672	Standard

Table D.2: Exact tests CRP

D.1.2. Heuristic CRP test results

Problem type	Settings	Horizon length	Components	Heuristic solution	Exact solution	Heuristic solved
Inout flow	Double block	1	4	16	16	Yes
Inout flow	Double block	2	4	16	16	Yes
Inout flow	Low storage	1	4	16	16	Yes
Inout flow	Low storage	2	4	16	16	Yes
Inout flow	Standard	1	4	16	16	Yes
Inout flow	Standard	2	4	16	16	Yes
Inout flow	No-block	1	4	16	16	Yes
Inout flow	No-block	2	4	16	16	Yes
Inout flow	Cluster	1	4	16	16	Yes
Inout flow	Cluster	2	4	16	16	Yes
Combined	Double block	1	18	0	64	No
Combined	Double block	2	18	0	64	No
Combined	Low storage	1	18	0	64	No
Combined	Low storage	2	18	0	64	No
Combined	Standard	1	18	0	64	No
Combined	Standard	2	18	0	64	No
Combined	No-block	1	18	0	64	No
Combined	No-block	2	18	0	64	No
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Settings	Horizon length	Components	Heuristic solution	Exact solution	Heuristic solved
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Combined	Cluster	1	18	0	64	No
Combined	Cluster	2	18	64	64	Yes
Out flow	Double block	1	4	10	10	Yes
Out flow	Double block	2	4	10	10	Yes
Out flow	Low storage	1	4	10	10	Yes
Out flow	Low storage	2	4	10	10	Yes
Out flow	Standard	1	4	10	10	Yes
Out flow	Standard	2	4	10	10	Yes
Out flow	Double block	1	4	8	8	Yes
Out flow	Double block	2	4	8	8	Yes
Out flow	Low storage	1	4	8	8	Yes
Out flow	Low storage	2	4	8	8	Yes
Out flow	Standard	1	4	8	8	Yes
Out flow	Standard	2	4	8	8	Yes
Out flow	Low storage	1	4	8	8	Yes
Out flow	Low storage	2	4	8	8	Yes
Out flow	Double block	1	4	8	8	Yes
Out flow	Double block	2	4	8	8	Yes
Out flow	Standard	1	4	8	8	Yes
Out flow	Standard	2	4	8	8	Yes
Combined	No-block	1	6	16	16	Yes
Combined	No-block	2	6	16	16	Yes
Combined	Low storage	1	6	16	16	Yes
Combined	Low storage	2	6	16	16	Yes
Combined	Double block	1	6	16	16	Yes
Combined	Double block	2	6	16	16	Yes
Combined	Standard	1	6	16	16	Yes
Combined	Standard	2	6	16	16	Yes
Out flow	Low storage	1	4	8	8	Yes
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Settings	Horizon length	Components	Heuristic solution	Exact solution	Heuristic solved
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Out flow	Low storage	2	4	8	8	Yes
Out flow	Double block	1	4	8	8	Yes
Out flow	Double block	2	4	8	8	Yes
Out flow	Standard	1	4	8	8	Yes
Out flow	Standard	2	4	8	8	Yes
Inout flow	No-block	1	12	46	46	Yes
Inout flow	No-block	2	12	46	46	Yes
Inout flow	Low storage	1	12	46	46	Yes
Inout flow	Low storage	2	12	46	46	Yes
Inout flow	Double block	1	12	46	46	Yes
Inout flow	Double block	2	12	46	46	Yes
Inout flow	Standard	1	12	46	46	Yes
Inout flow	Standard	2	12	46	46	Yes
Inout flow	No-block	1	6	24	24	Yes
Inout flow	No-block	2	6	24	24	Yes
Inout flow	Low storage	1	6	24	24	Yes
Inout flow	Low storage	2	6	24	24	Yes
Inout flow	Double block	1	6	24	24	Yes
Inout flow	Double block	2	6	24	24	Yes
Inout flow	Standard	1	6	24	24	Yes
Inout flow	Standard	2	6	24	24	Yes

Table D.3: CRP heuristic results

D.1.3. Component relocation result Walney

The best solution found for the component relocation problem in the walney project is given. First the locations are given in Figure D.1, and subsequently the relocation schedule is given.

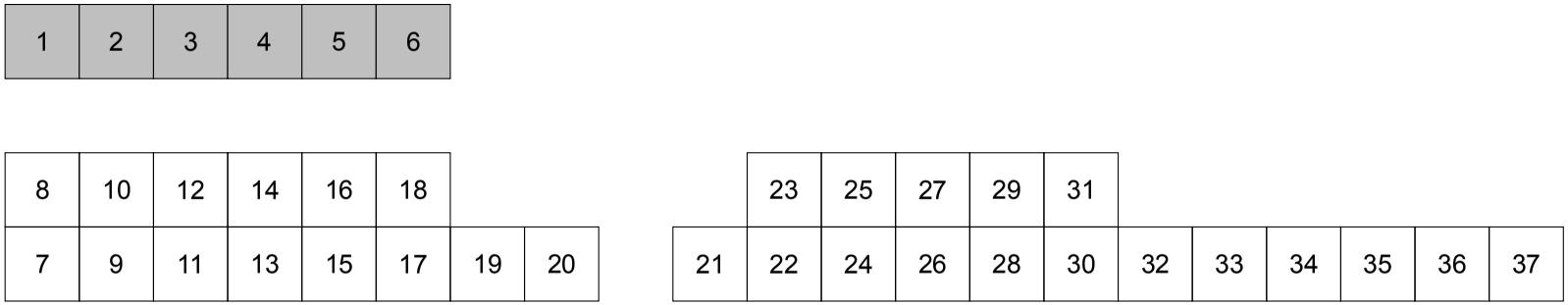


Figure D.1: Layout of Walney TP storage. Loadout locations are in gray. Positions 0 and 38 denote the arrival and departure locations, respectively

Block	From	To
G04	0	1
D03	0	2
E02	0	3
C05	0	4
D02	0	5
F01	0	6
G04	1	17
D03	2	15
E02	3	13
C05	4	11
D02	5	16
F01	6	14
B04	0	1
C04	0	2
B05	0	3
C03	0	4
E01	0	5
C02	0	6
B04	1	24
C04	2	22
B05	3	21
C03	4	19
E01	5	20
C02	6	18
A01	0	1
A02	0	2
A03	0	3
B01	0	4
D01	0	5
B02	0	6
A01	1	35
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
A02	2	34
A03	3	33
B01	4	26
D01	5	28
B02	6	27
C01	0	1
B03	0	2
A05	0	3
A04	0	4
F02	0	5
E03	0	6
C01	1	36
B03	2	37
A05	3	7
A04	4	29
F02	5	25
E03	6	12
E04	0	4
H01	0	5
H04	0	3
F03	0	2
C06	0	1
G01	0	6
G01	6	8
E03	12	6
E03	6	38
C06	1	38
C05	11	1
H01	5	30
E04	4	9
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
G02	0	5
D04	0	6
D04	6	31
G02	5	23
H04	3	10
H02	0	3
C07	0	4
D05	0	5
G03	0	6
C05	1	38
F03	2	38
F01	14	2
D04	31	1
D04	1	38
F01	2	38
G02	23	2
E02	13	1
E02	1	38
G02	2	38
D02	16	1
G03	6	11
D03	15	6
D03	6	38
D02	1	38
D05	5	15
H03	0	2
H05	0	1
F05	0	5
F04	0	6
H05	1	38
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
H03	2	38
B05	21	2
F02	25	1
F02	1	38
B05	2	38
A06	0	2
B06	0	1
B06	1	31
A06	2	25
C04	22	1
H02	3	22
C07	4	21
G01	8	2
F05	5	16
F04	6	14
H06	0	3
G05	0	4
F06	0	5
A08	0	6
C04	1	38
G01	2	38
A06	25	8
C07	21	2
H02	22	21
B04	24	1
B06	31	24
A06	8	25
A08	6	23
H01	30	6
B04	1	38
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
H01	6	38
E01	20	6
C02	18	1
C02	1	38
E01	6	38
A05	7	1
F06	5	7
C03	19	5
A04	29	6
A04	6	38
C03	5	38
G05	4	19
D01	28	5
D01	5	38
A05	1	38
H06	3	20
C07	2	8
D07	0	2
B08	0	3
B07	0	4
C08	0	1
E05	0	5
D06	0	6
D06	6	30
A03	33	6
C08	1	32
F04	14	1
A03	6	38
F04	1	38
A02	34	6
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
G03	11	1
A02	6	38
G03	1	38
A01	35	6
H02	21	1
H02	1	38
A01	6	38
B03	37	1
B02	27	6
B02	6	38
B03	1	38
E05	5	21
F05	16	5
D05	15	6
D07	2	13
B08	3	37
B07	4	31
G04	17	2
C01	36	1
B01	26	3
C01	1	38
B01	3	38
H04	10	1
E04	9	3
F05	5	38
E04	3	38
H04	1	38
G04	2	38
D05	6	38
C09	0	1
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
D09	0	2
E08	0	3
B09	0	4
D08	0	5
C10	0	6
C10	6	33
D08	5	34
B09	4	26
E08	3	17
D09	2	11
C09	1	18
H06	20	1
C07	8	5
E05	21	4
F06	7	3
G05	19	6
H06	1	38
F06	3	38
G05	6	38
E05	4	38
C07	5	38
E07	0	1
B10	0	2
B11	0	3
E06	0	4
F07	0	5
A07	0	6
A07	6	21
F07	5	19
E06	4	20
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
B11	3	9
B10	2	7
E07	1	12
D07	13	6
D08	34	2
E06	20	5
E07	12	4
F07	19	1
D07	6	38
E07	4	38
D08	2	38
F07	1	38
E06	5	38
A10	0	1
A11	0	2
A12	0	3
B12	0	4
A09	0	5
B13	0	6
B13	6	36
A09	5	20
B12	4	15
A12	3	16
A11	2	10
A10	1	8
C09	18	3
E08	17	5
C10	33	2
B09	26	6
D09	11	1
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
B09	6	38
D09	1	38
E08	5	38
C09	3	38
C10	2	38
A06	25	5
C08	32	6
B07	31	2
B06	24	3
D06	30	1
D06	1	38
B06	3	38
A06	5	38
C08	6	38
B07	2	38
B17	0	4
A14	0	5
A13	0	6
B14	0	3
B16	0	2
B15	0	1
B15	1	13
A13	6	33
A14	5	32
B17	4	26
A10	8	5
B08	37	1
A07	21	6
B14	3	30
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
B16	2	37
A09	20	2
A08	23	4
A09	2	38
A08	4	38
A07	6	38
B08	1	38
A10	5	38
C17	0	1
C21	0	2
C15	0	3
C12	0	4
C20	0	5
C19	0	6
C19	6	22
C20	5	21
C12	4	17
C15	3	19
C21	2	20
C17	1	18
A13	33	3
A11	10	6
A12	16	4
B15	13	2
A14	32	1
A12	4	38
A14	1	38
A13	3	38
A11	6	38
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
B15	2	38
C20	21	1
C21	20	6
C19	22	5
B17	26	2
B16	37	3
C20	1	38
B17	2	38
B16	3	38
C19	5	38
C21	6	38
C17	18	5
C11	0	4
C13	0	6
C16	0	3
C14	0	2
C18	0	1
C18	1	38
C14	2	38
C16	3	38
C13	6	38
C17	5	38
B11	9	2
B12	15	5
C12	17	1
B10	7	6
C12	1	38
B10	6	38
C11	4	38
⋮	⋮	⋮

Block	From	To
⋮	⋮	⋮
B12	5	38
B11	2	38
C15	19	3
B13	36	6
B14	30	2
B14	2	38
C15	3	38
B13	6	38

D.2. Routing tests

The routing tests were performed for three different fields, shown in Appendix D.2.1. The ships used were the same as those in the walney environment, although with some modifications. The details are given in Table D.4.

Installation methods	All turbines are installable by the combined method, and 25% randomly picked turbines are installable by the Aeolus-only method.
Plugs:	8 Plugs. For both the top as bottom plugs, 2 plug types are used. For each type 2 plugs were available.
Arrival	When component arrival is considered, transition pieces arrive in batches of 4 with an interval of 48 hours.

Table D.4: Details of routing tests

D.2.1. Fields

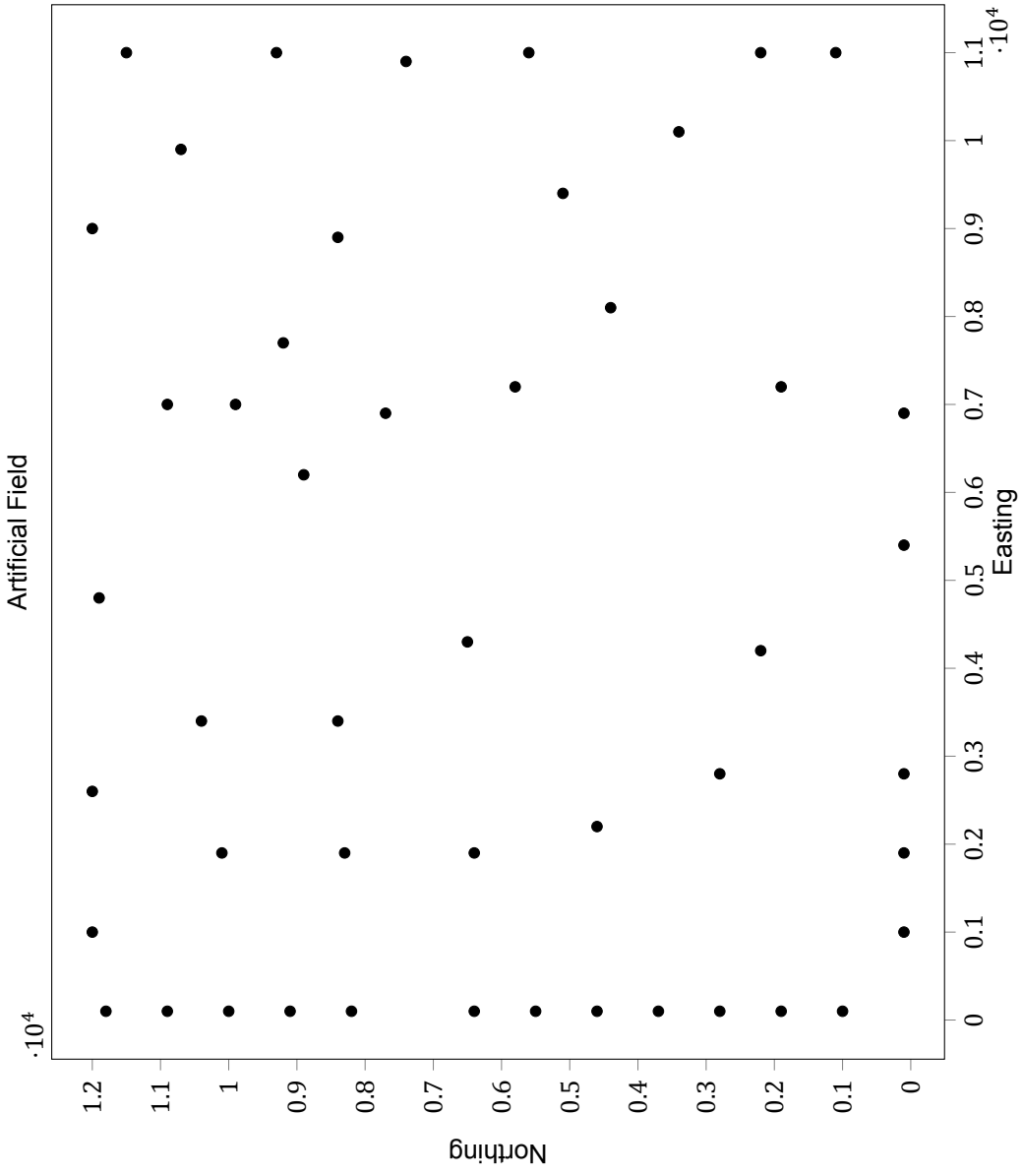


Figure D.2: Turbines in the Artificial field

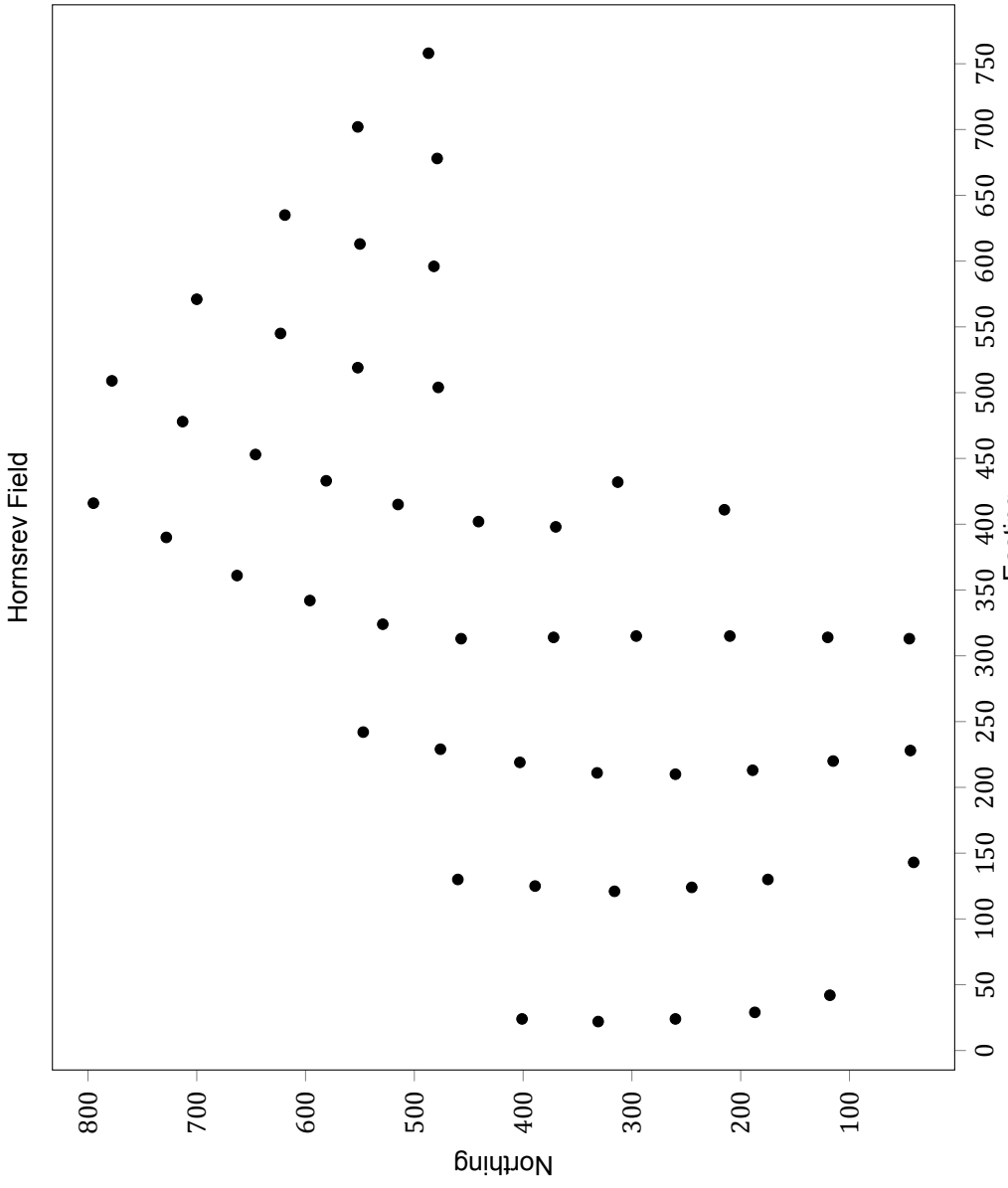


Figure D.3: Turbines in the Hornsrev field

D.2.2. Routing exact tests without cutting planes

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
Blank	Walney	6	313	4579991	62450311	No	0	0.22
Precedence	Walney	6	366	778264	14358663	Yes	10	0.00
Plugs	Walney	6	366	6338666	6904186	No	10	0.21
Arrival	Walney	6	369	4767720	17519465	No	10	0.11
Blank	Hornsrev	6	314	4341079	63459555	No	0	0.16
Precedence	Hornsrev	6	368	1161040	21061139	Yes	10	0.00
Plugs	Hornsrev	6	368	6570279	8106069	No	10	0.16
Arrival	Hornsrev	6	369	5111942	17919856	No	10	0.13
Blank	Optimized	6	317	5191185	56932619	No	0	0.22
Precedence	Optimized	6	369	736875	13988274	Yes	10	0.00
Plugs	Optimized	6	366	6081116	11980518	No	10	0.15
Arrival	Optimized	6	370	5814966	12213594	No	10	0.12

D.2.3. Routing exact tests with cutting planes

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
Blank	Waiting arc	5	278	429681	10413176	Yes	0.00	0.00
Blank	K flow	5	278	130398	4557706	Yes	0.00	0.00
Blank	None	5	278	156002	5300136	Yes	0.00	0.00
Blank	Waiting node	5	278	442550	7557642	Yes	0.00	0.00
Blank	Trips-back	5	278	386885	8939473	Yes	0.00	0.00
Blank	Min routes	5	278	327688	9793577	Yes	0.00	0.00
Blank	Trip symmetry	5	278	70784	2463960	Yes	0.00	0.00
Blank	Kr demand	5	278	637992	15846114	Yes	0.00	0.00
Blank	K demand	5	278	226943	6979131	Yes	0.00	0.00
Blank	Kr flow	5	278	117879	4080239	Yes	0.00	0.00
Blank	Demand	5	278	332866	8085648	Yes	0.00	0.00
Blank	Full trips	5	278	156002	5300136	Yes	0.00	0.00
Blank	Total-time	5	278	148	155	Yes	248.70	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Precedence	Waiting arc	5	327	81781	1661430	Yes	5.10	0.00
Precedence	K flow	5	327	9838	257637	Yes	5.10	0.00
Precedence	None	5	327	166317	2756906	Yes	5.10	0.00
Precedence	Waiting node	5	327	74122	1056117	Yes	5.10	0.00
Precedence	Trips-back	5	327	27365	902186	Yes	5.10	0.00
Precedence	Min routes	5	327	138563	2179331	Yes	5.10	0.00
Precedence	Trip symmetry	5	327	26626	713031	Yes	5.10	0.00
Precedence	Kr demand	5	327	47049	824159	Yes	5.10	0.00
Precedence	K demand	5	327	101825	2362559	Yes	5.10	0.00
Precedence	Kr flow	5	327	8568	259272	Yes	5.10	0.00
Precedence	Demand	5	327	155352	2103972	Yes	5.10	0.00
Precedence	Full trips	5	327	166317	2756906	Yes	5.10	0.00
Precedence	Total-time	5	327	1037	54081	Yes	248.68	0.00
Plugs	Waiting arc	5	327	767018	5527840	Yes	5.10	0.00
Plugs	K flow	5	327	93446	352901	Yes	5.10	0.00
Plugs	None	5	327	1311177	24958275	No	5.10	0.18
Plugs	Waiting node	5	327	632249	2774989	Yes	5.10	0.00
Plugs	Trips-back	5	327	1586085	4692045	No	5.10	0.23
Plugs	Min routes	5	327	451901	1964974	Yes	5.10	0.00
Plugs	Trip symmetry	5	327	483815	1743213	Yes	5.10	0.00
Plugs	Kr demand	5	327	726164	6112822	Yes	5.10	0.00
Plugs	K demand	5	327	974735	2274243	Yes	5.10	0.00
Plugs	Kr flow	5	327	181208	329470	Yes	5.10	0.00
Plugs	Demand	5	327	622891	2348484	Yes	5.10	0.00
Plugs	Full trips	5	327	1352383	25239591	No	5.10	0.18
Plugs	Total-time	5	327	314938	2320283	Yes	184.89	0.00
Arrival	Waiting arc	5	326	1570392	6528094	No	6.86	0.02
Arrival	K flow	5	326	138731	408074	Yes	6.86	0.00
Arrival	None	5	326	1629072	7452235	No	6.86	0.21
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Arrival	Waiting node	5	326	417658	2395882	Yes	6.86	0.00
Arrival	Trips-back	5	326	1188563	10193896	Yes	6.86	0.00
Arrival	Min routes	5	326	47267	485849	Yes	6.86	0.00
Arrival	Trip symmetry	5	326	751417	5030125	Yes	6.86	0.00
Arrival	Kr demand	5	326	545571	2125277	Yes	6.86	0.00
Arrival	K demand	5	326	1151112	4196026	Yes	6.86	0.00
Arrival	Kr flow	5	326	108215	331595	Yes	6.86	0.00
Arrival	Demand	5	326	1650298	3726937	No	6.86	0.24
Arrival	Full trips	5	326	1628784	7448691	No	6.86	0.21
Arrival	Total-time	5	326	11452	87334	Yes	184.89	0.00
Blank	Waiting arc	5	276	312400	7975131	Yes	0.00	0.00
Blank	K flow	5	276	108155	4281903	Yes	0.00	0.00
Blank	None	5	276	470869	13958524	Yes	0.00	0.00
Blank	Waiting node	5	276	216863	6176283	Yes	0.00	0.00
Blank	Trips-back	5	276	405176	9650707	Yes	0.00	0.00
Blank	Min routes	5	276	300223	7970672	Yes	0.00	0.00
Blank	Trip symmetry	5	276	72586	2279034	Yes	0.00	0.00
Blank	Kr demand	5	276	391340	13595580	Yes	0.00	0.00
Blank	K demand	5	276	556213	16030607	Yes	0.00	0.00
Blank	Kr flow	5	276	162834	3762940	Yes	0.00	0.00
Blank	Demand	5	276	207459	5425915	Yes	0.00	0.00
Blank	Full trips	5	276	470869	13958524	Yes	0.00	0.00
Blank	Total-time	5	276	97	198	Yes	247.19	0.00
Precedence	Waiting arc	5	326	102947	2090508	Yes	5.10	0.00
Precedence	K flow	5	326	9227	293005	Yes	5.10	0.00
Precedence	None	5	326	78140	1894302	Yes	5.10	0.00
Precedence	Waiting node	5	326	61604	1458050	Yes	5.10	0.00
Precedence	Trips-back	5	326	20616	805938	Yes	5.10	0.00
Precedence	Min routes	5	326	85905	1369078	Yes	5.10	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Precedence	Trip symmetry	5	326	63979	1098159	Yes	5.10	0.00
Precedence	Kr demand	5	326	136724	1825225	Yes	5.10	0.00
Precedence	K demand	5	326	103987	2347660	Yes	5.10	0.00
Precedence	Kr flow	5	326	12548	272191	Yes	5.10	0.00
Precedence	Demand	5	326	166791	2174921	Yes	5.10	0.00
Precedence	Full trips	5	326	78140	1894302	Yes	5.10	0.00
Precedence	Total-time	5	326	1123	66057	Yes	247.11	0.00
Plugs	Waiting arc	5	325	972993	2405750	Yes	5.10	0.00
Plugs	K flow	5	325	166590	320658	Yes	5.10	0.00
Plugs	None	5	325	331820	1735441	Yes	5.10	0.00
Plugs	Waiting node	5	325	1680754	3385974	No	5.10	0.23
Plugs	Trips-back	5	325	1630163	2217375	No	5.10	0.21
Plugs	Min routes	5	325	1611971	4316547	No	5.10	0.37
Plugs	Trip symmetry	5	325	1519719	5374967	No	5.10	0.22
Plugs	Kr demand	5	325	1537494	6214059	No	5.10	0.08
Plugs	K demand	5	325	1583474	2913434	No	5.10	0.18
Plugs	Kr flow	5	325	216657	339589	Yes	5.10	0.00
Plugs	Demand	5	325	782188	3761817	Yes	5.10	0.00
Plugs	Full trips	5	325	331820	1735441	Yes	5.10	0.00
Plugs	Total-time	5	325	358910	2543408	Yes	184.87	0.00
Arrival	Waiting arc	5	325	1003160	6506510	Yes	6.86	0.00
Arrival	K flow	5	325	1569382	2871786	No	6.86	0.15
Arrival	None	5	325	1590125	10018207	No	6.86	0.03
Arrival	Waiting node	5	325	1347474	8124778	Yes	6.86	0.00
Arrival	Trips-back	5	325	132037	917237	Yes	6.86	0.00
Arrival	Min routes	5	325	376613	1203284	Yes	6.86	0.00
Arrival	Trip symmetry	5	325	1637166	7549987	No	6.86	0.19
Arrival	Kr demand	5	325	1644436	4888560	No	6.86	0.16
Arrival	K demand	5	325	1623061	3780032	No	6.86	0.21
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Arrival	Kr flow	5	325	96582	496161	Yes	6.86	0.00
Arrival	Demand	5	325	428898	1007667	Yes	6.86	0.00
Arrival	Full trips	5	325	1591922	10025421	No	6.86	0.03
Arrival	Total-time	5	325	1510194	10380213	No	185.15	0.00
Blank	Waiting arc	5	279	215098	6635189	Yes	0.00	0.00
Blank	K flow	5	279	117849	4141078	Yes	0.00	0.00
Blank	None	5	279	369332	9325427	Yes	0.00	0.00
Blank	Waiting node	5	279	355943	9910499	Yes	0.00	0.00
Blank	Trips-back	5	279	282852	9470049	Yes	0.00	0.00
Blank	Min routes	5	279	757172	13667045	Yes	0.00	0.00
Blank	Trip symmetry	5	279	122472	4366155	Yes	0.00	0.00
Blank	Kr demand	5	279	378775	11599876	Yes	0.00	0.00
Blank	K demand	5	279	428604	12584705	Yes	0.00	0.00
Blank	Kr flow	5	279	148700	4898884	Yes	0.00	0.00
Blank	Demand	5	279	250124	6642846	Yes	0.00	0.00
Blank	Full trips	5	279	369332	9325427	Yes	0.00	0.00
Blank	Total-time	5	279	113	10	Yes	250.10	0.00
Precedence	Waiting arc	5	326	109887	2136557	Yes	5.10	0.00
Precedence	K flow	5	326	10481	323665	Yes	5.10	0.00
Precedence	None	5	326	112051	1714365	Yes	5.10	0.00
Precedence	Waiting node	5	326	79134	1900782	Yes	5.10	0.00
Precedence	Trips-back	5	326	15974	696363	Yes	5.10	0.00
Precedence	Min routes	5	326	166773	2673734	Yes	5.10	0.00
Precedence	Trip symmetry	5	326	27306	740784	Yes	5.10	0.00
Precedence	Kr demand	5	326	80803	1793127	Yes	5.10	0.00
Precedence	K demand	5	326	139508	1753279	Yes	5.10	0.00
Precedence	Kr flow	5	326	8061	296699	Yes	5.10	0.00
Precedence	Demand	5	326	87626	1771736	Yes	5.10	0.00
Precedence	Full trips	5	326	112051	1714365	Yes	5.10	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Precedence	Total-time	5	326	917	56121	Yes	248.44	0.00
Plugs	Waiting arc	5	327	564627	1668001	Yes	5.10	0.00
Plugs	K flow	5	327	247971	488309	Yes	5.10	0.00
Plugs	None	5	327	510010	3739042	Yes	5.10	0.00
Plugs	Waiting node	5	327	1671890	2532447	No	5.10	0.24
Plugs	Trips-back	5	327	367345	1741050	Yes	5.10	0.00
Plugs	Min routes	5	327	1336715	16281876	No	5.10	0.24
Plugs	Trip symmetry	5	327	246200	2115710	Yes	5.10	0.00
Plugs	Kr demand	5	327	1606384	2750810	No	5.10	0.23
Plugs	K demand	5	327	1636290	4946888	No	5.10	0.14
Plugs	Kr flow	5	327	301132	298746	Yes	5.10	0.00
Plugs	Demand	5	327	1586442	7508416	No	5.10	0.19
Plugs	Full trips	5	327	510010	3739042	Yes	5.10	0.00
Plugs	Total-time	5	327	620865	2769631	Yes	186.82	0.00
Arrival	Waiting arc	5	326	499784	2982874	Yes	6.86	0.00
Arrival	K flow	5	326	61908	722580	Yes	6.86	0.00
Arrival	None	5	326	482065	1975745	Yes	6.86	0.00
Arrival	Waiting node	5	326	1669877	5350510	No	6.86	0.57
Arrival	Trips-back	5	326	1487438	8980902	No	6.86	0.22
Arrival	Min routes	5	326	798156	4305177	Yes	6.86	0.00
Arrival	Trip symmetry	5	326	240186	810832	Yes	6.86	0.00
Arrival	Kr demand	5	326	1601730	6259247	No	6.86	0.13
Arrival	K demand	5	326	889448	5630427	Yes	6.86	0.00
Arrival	Kr flow	5	326	187409	476519	Yes	6.86	0.00
Arrival	Demand	5	326	1551298	3374338	No	6.86	0.15
Arrival	Full trips	5	326	482065	1975745	Yes	6.86	0.00
Arrival	Total-time	5	326	1551217	5048528	Yes	186.18	0.00

D.2.4. Routing exact tests with standard

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
Blank	Kr demand	18	849	270933	124393	Yes	838.40	0.00
Blank	Demand	18	849	142635	133168	Yes	838.25	0.00
Blank	K flow	18	849	18075	8834	Yes	839.03	0.00
Blank	Waiting node	18	849	115195	188756	Yes	838.17	0.00
Blank	Standard	18	849	62408	141180	Yes	838.17	0.00
Blank	K demand	18	849	188793	111388	Yes	838.40	0.00
Blank	Kr flow	18	849	7982	3968	Yes	839.03	0.00
Precedence	Kr demand	8	437	5528	134060	Yes	399.43	0.00
Precedence	Demand	8	437	6674	147350	Yes	399.34	0.00
Precedence	K flow	8	437	5745	103633	Yes	399.49	0.00
Precedence	Waiting node	8	437	5573	128463	Yes	399.28	0.00
Precedence	Standard	8	437	4666	137120	Yes	399.28	0.00
Precedence	K demand	8	437	6265	151821	Yes	399.43	0.00
Precedence	Kr flow	8	437	6377	105184	Yes	399.49	0.00
Plugs	Kr demand	6	369	34657	130328	Yes	248.68	0.00
Plugs	Demand	6	369	1417036	3915680	No	248.48	0.03
Plugs	K flow	6	369	30182	85220	Yes	248.84	0.00
Plugs	Waiting node	6	369	1283312	3992978	No	248.48	0.03
Plugs	Standard	6	369	14636	63696	Yes	248.48	0.00
Plugs	K demand	6	369	1217235	2325390	No	248.68	0.03
Plugs	Kr flow	6	369	31381	115131	Yes	248.84	0.00
Arrival	Kr demand	6	368	22704	105247	Yes	248.25	0.00
Arrival	Demand	6	368	1258742	6146446	Yes	248.15	0.00
Arrival	K flow	6	368	20997	108999	Yes	248.28	0.00
Arrival	Waiting node	6	368	971477	2772992	Yes	248.15	0.00
Arrival	Standard	6	368	171990	1126251	Yes	248.15	0.00
Arrival	K demand	6	368	78910	424756	Yes	248.25	0.00
Arrival	Kr flow	6	368	31234	72097	Yes	248.28	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Blank	Kr demand	18	851	1406915	508783	No	831.67	0.00
Blank	Demand	18	851	466622	216421	Yes	832.34	0.00
Blank	K flow	18	851	41366	12530	Yes	840.69	0.00
Blank	Waiting node	18	851	388534	275250	Yes	831.21	0.00
Blank	Standard	18	851	143248	191619	Yes	831.21	0.00
Blank	K demand	18	851	923534	275431	Yes	831.67	0.00
Blank	Kr flow	18	851	20032	5056	Yes	840.69	0.00
Precedence	Kr demand	8	437	5336	121192	Yes	396.79	0.00
Precedence	Demand	8	437	7630	143129	Yes	396.65	0.00
Precedence	K flow	8	437	6771	98911	Yes	399.91	0.00
Precedence	Waiting node	8	437	5816	109433	Yes	396.58	0.00
Precedence	Standard	8	437	5120	133727	Yes	396.58	0.00
Precedence	K demand	8	437	5944	142359	Yes	396.79	0.00
Precedence	Kr flow	8	437	10141	98598	Yes	399.91	0.00
Plugs	Kr demand	6	367	76334	352461	Yes	246.42	0.00
Plugs	Demand	6	367	28629	111946	Yes	246.42	0.00
Plugs	K flow	6	367	27953	80020	Yes	246.65	0.00
Plugs	Waiting node	6	367	152389	487813	Yes	246.42	0.00
Plugs	Standard	6	367	96599	377812	Yes	246.42	0.00
Plugs	K demand	6	367	161696	821932	Yes	246.42	0.00
Plugs	Kr flow	6	367	43304	60415	Yes	246.65	0.00
Arrival	Kr demand	6	368	147116	879814	Yes	245.77	0.00
Arrival	Demand	6	368	198663	1675520	Yes	245.74	0.00
Arrival	K flow	6	368	34421	82274	Yes	246.38	0.00
Arrival	Waiting node	6	368	98433	294558	Yes	245.74	0.00
Arrival	Standard	6	368	86351	472050	Yes	245.74	0.00
Arrival	K demand	6	368	194444	674029	Yes	245.77	0.00
Arrival	Kr flow	6	368	39993	163299	Yes	246.38	0.00
Blank	Kr demand	18	848	1502604	1028696	No	828.40	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Solution	Det time	Nodes	Solved	Integer relaxation	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Blank	Demand	18	848	440619	401140	Yes	829.16	0.00
Blank	K flow	18	848	78654	23979	Yes	837.06	0.00
Blank	Waiting node	18	848	289921	429430	Yes	828.01	0.00
Blank	Standard	18	848	229913	409249	Yes	828.01	0.00
Blank	K demand	18	848	885440	608845	Yes	828.40	0.00
Blank	Kr flow	18	848	60609	18552	Yes	837.06	0.00
Precedence	Kr demand	8	433	5532	134645	Yes	394.60	0.00
Precedence	Demand	8	433	10909	191417	Yes	394.52	0.00
Precedence	K flow	8	433	6658	100727	Yes	396.11	0.00
Precedence	Waiting node	8	433	6893	110637	Yes	394.49	0.00
Precedence	Standard	8	433	4754	133640	Yes	394.49	0.00
Precedence	K demand	8	433	7524	121530	Yes	394.60	0.00
Precedence	Kr flow	8	433	9493	100144	Yes	396.11	0.00
Plugs	Kr demand	6	367	26704	99861	Yes	246.05	0.00
Plugs	Demand	6	367	103624	736267	Yes	245.97	0.00
Plugs	K flow	6	367	40625	109012	Yes	246.55	0.00
Plugs	Waiting node	6	367	775119	1657241	Yes	245.97	0.00
Plugs	Standard	6	367	23902	128040	Yes	245.97	0.00
Plugs	K demand	6	367	514694	1304030	Yes	246.05	0.00
Plugs	Kr flow	6	367	30302	115148	Yes	246.55	0.00
Arrival	Kr demand	6	366	45692	147590	Yes	246.26	0.00
Arrival	Demand	6	366	49731	255915	Yes	246.24	0.00
Arrival	K flow	6	366	9497	38335	Yes	246.45	0.00
Arrival	Waiting node	6	366	69114	229840	Yes	246.24	0.00
Arrival	Standard	6	366	64322	341347	Yes	246.24	0.00
Arrival	K demand	6	366	106741	289419	Yes	246.26	0.00
Arrival	Kr flow	6	366	31101	114114	Yes	246.45	0.00

D.2.5. Routing Branch Cut

Problem type	Cuts	Turbines	Sol.	Det time	Nodes	Solved	Int. relax.	Gap	Bc det time	Generated cuts
Blank	Standard	9	458	6824	5748	Yes	427.10	0.00	0	0
Blank	Branch cut 2 k	9	458	1670	171	Yes	429.70	0.00	0	0
Blank	Branch cut 1	9	458	376	0	Yes	427.10	0.00	0	9
Blank	Branch cut 1 k	9	458	1670	171	Yes	429.70	0.00	0	0
Blank	Branch cut 2	9	458	328	0	Yes	427.10	0.00	0	7
Blank	K flow	9	458	1670	171	Yes	429.70	0.00	0	0
Precedence	Standard	6	366	798566	708587	Yes	298.84	0.00	0	0
Precedence	Branch cut 2 k	6	366	5150	5681	Yes	302.27	0.00	0	0
Precedence	Branch cut 1	6	366	65929	55766	Yes	298.84	0.00	0	67
Precedence	Branch cut 1 k	6	366	5150	5681	Yes	302.27	0.00	0	0
Precedence	Branch cut 2	6	370	2642	3029	Yes	298.84	0.00	0	56
Precedence	K flow	6	366	5150	5681	Yes	302.27	0.00	0	0
Plugs	Standard	5	332	48492	22684	Yes	213.56	0.00	0	0
Plugs	Branch cut 2 k	5	396	57665	19639	Yes	215.89	0.00	0	1
Plugs	Branch cut 1	5	332	11312	5890	Yes	213.56	0.00	0	57
Plugs	Branch cut 1 k	5	332	21747	5968	Yes	215.89	0.00	0	0
Plugs	Branch cut 2	5	356	6328	3216	Yes	213.56	0.00	0	89
Plugs	K flow	5	332	21747	5968	Yes	215.89	0.00	0	0
Arrival	Standard	5	332	572789	432154	No	214.39	0.03	0	0
Arrival	Branch cut 2 k	5	397	33493	13944	Yes	215.97	0.00	0	1
Arrival	Branch cut 1	5	332	34813	18124	Yes	214.39	0.00	0	93
Arrival	Branch cut 1 k	5	332	57760	11980	Yes	215.97	0.00	0	0
Arrival	Branch cut 2	5	338	3377	1550	Yes	214.39	0.00	0	64
Arrival	K flow	5	332	57760	11980	Yes	215.97	0.00	0	0
Blank	Standard	9	459	26135	20249	Yes	426.85	0.00	0	0
Blank	Branch cut 2 k	9	459	1308	146	Yes	430.59	0.00	0	0
Blank	Branch cut 1	9	459	661	99	Yes	426.85	0.00	0	11
Blank	Branch cut 1 k	9	459	1308	146	Yes	430.59	0.00	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Sol.	Det time	Nodes	Solved	Int. relax.	Gap	Bc det time	Generated cuts
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Blank	Branch cut 2	9	461	656	0	Yes	426.85	0.00	0	9
Blank	K flow	9	459	1308	146	Yes	430.59	0.00	0	0
Precedence	Standard	6	371	87226	95596	Yes	304.25	0.00	0	0
Precedence	Branch cut 2 k	6	371	4289	5202	Yes	308.59	0.00	0	0
Precedence	Branch cut 1	6	371	6283	6567	Yes	304.25	0.00	0	37
Precedence	Branch cut 1 k	6	371	4289	5202	Yes	308.59	0.00	0	0
Precedence	Branch cut 2	6	377	874	909	Yes	304.25	0.00	0	28
Precedence	K flow	6	371	4289	5202	Yes	308.59	0.00	0	0
Plugs	Standard	5	333	35019	23586	Yes	215.98	0.00	0	0
Plugs	Branch cut 2 k	5	398	61443	20057	Yes	217.95	0.00	0	1
Plugs	Branch cut 1	5	333	22300	7382	Yes	215.98	0.00	0	95
Plugs	Branch cut 1 k	5	333	41792	14080	Yes	217.95	0.00	0	0
Plugs	Branch cut 2	5	337	9886	4560	Yes	215.98	0.00	0	79
Plugs	K flow	5	333	41792	14080	Yes	217.95	0.00	0	0
Arrival	Standard	5	334	12201	9397	Yes	214.89	0.00	0	0
Arrival	Branch cut 2 k	5	398	11889	4688	Yes	216.02	0.00	0	1
Arrival	Branch cut 1	5	334	57275	17000	Yes	214.89	0.00	0	90
Arrival	Branch cut 1 k	5	334	50550	13066	Yes	216.02	0.00	0	0
Arrival	Branch cut 2	5	341	5353	2831	Yes	214.89	0.00	0	63
Arrival	K flow	5	334	50550	13066	Yes	216.02	0.00	0	0
Blank	Standard	9	461	3747	2499	Yes	431.15	0.00	0	0
Blank	Branch cut 2 k	9	461	1022	1	Yes	431.74	0.00	0	0
Blank	Branch cut 1	9	461	1439	246	Yes	431.15	0.00	0	28
Blank	Branch cut 1 k	9	461	1022	1	Yes	431.74	0.00	0	0
Blank	Branch cut 2	9	463	3881	1008	Yes	431.15	0.00	0	26
Blank	K flow	9	461	1022	1	Yes	431.74	0.00	0	0
Precedence	Standard	6	368	5507	6612	Yes	302.68	0.00	0	0
Precedence	Branch cut 2 k	6	368	4158	5141	Yes	304.81	0.00	0	0
Precedence	Branch cut 1	6	368	10513	12507	Yes	302.68	0.00	0	50
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Problem type	Cuts	Turbines	Sol.	Det time	Nodes	Solved	Int. relax.	Gap	Bc det time	Generated cuts
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Precedence	Branch cut 1 k	6	368	4158	5141	Yes	304.81	0.00	0	0
Precedence	Branch cut 2	6	368	931	805	Yes	302.68	0.00	0	15
Precedence	K flow	6	368	4158	5141	Yes	304.81	0.00	0	0
Plugs	Standard	5	334	21341	11764	Yes	216.78	0.00	0	0
Plugs	Branch cut 2 k	5	399	609535	153366	No	217.33	0.09	0	1
Plugs	Branch cut 1	5	334	41681	16338	Yes	216.78	0.00	0	99
Plugs	Branch cut 1 k	5	334	206624	51400	Yes	217.33	0.00	0	0
Plugs	Branch cut 2	5	344	5700	3214	Yes	216.78	0.00	0	88
Plugs	K flow	5	334	206624	51400	Yes	217.33	0.00	0	0
Arrival	Standard	5	335	14247	10000	Yes	217.72	0.00	0	0
Arrival	Branch cut 2 k	5	400	33536	12103	Yes	218.15	0.00	0	1
Arrival	Branch cut 1	5	335	31180	23702	Yes	217.72	0.00	0	39
Arrival	Branch cut 1 k	5	335	69418	22801	Yes	218.15	0.00	0	0
Arrival	Branch cut 2	5	345	3767	2168	Yes	217.72	0.00	0	70
Arrival	K flow	5	335	69418	22801	Yes	218.15	0.00	0	0

D.2.6. Routing Heuristic

Type	Settings	Heuristic solution	Exact solution	Duration	Turbines	Field	Gap
Blank	Repair broken	356.52	356.52	501.48	6	Walney	0.00
Blank	Repair broken	356.52	356.52	500.28	6	Walney	0.00
Blank	Repair broken	356.52	356.52	499.79	6	Walney	0.00
Blank	Repair waiting	356.52	356.52	502.20	6	Walney	0.00
Blank	Repair waiting	356.52	356.52	499.77	6	Walney	0.00
Blank	Repair waiting	356.52	356.52	503.11	6	Walney	0.00
Precedence	Repair broken	366.53	366.53	583.97	6	Walney	0.00
Precedence	Repair broken	366.53	366.53	582.17	6	Walney	0.00
Precedence	Repair broken	366.53	366.53	579.49	6	Walney	0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Settings	Heuristic solution	Exact solution	Duration	Turbines	Field	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Precedence	Repair waiting	366.53	366.53	583.55	6	Walney	0.00
Precedence	Repair waiting	366.53	366.53	590.16	6	Walney	0.00
Precedence	Repair waiting	366.53	366.53	593.22	6	Walney	0.00
Plugs	Repair broken	374.59	366.96	535.55	6	Walney	2.08
Plugs	Repair broken	366.96	366.96	564.89	6	Walney	0.00
Plugs	Repair broken	366.96	366.96	567.91	6	Walney	0.00
Plugs	Repair waiting	367.15	366.96	612.82	6	Walney	0.05
Plugs	Repair waiting	366.96	366.96	634.09	6	Walney	0.00
Plugs	Repair waiting	366.96	366.96	607.21	6	Walney	0.00
Arrival	Repair broken	366.43	366.43	532.53	6	Walney	0.00
Arrival	Repair broken	366.61	366.43	550.06	6	Walney	0.05
Arrival	Repair broken	374.55	366.43	529.58	6	Walney	2.22
Arrival	Repair waiting	366.82	366.43	589.99	6	Walney	0.11
Arrival	Repair waiting	366.61	366.43	612.95	6	Walney	0.05
Arrival	Repair waiting	366.48	366.43	589.10	6	Walney	0.01
Blank	Repair broken	357.11	357.11	501.15	6	Generated	0.00
Blank	Repair broken	357.11	357.11	499.90	6	Generated	0.00
Blank	Repair broken	357.11	357.11	497.78	6	Generated	0.00
Blank	Repair waiting	357.11	357.11	498.13	6	Generated	0.00
Blank	Repair waiting	357.11	357.11	500.90	6	Generated	0.00
Blank	Repair waiting	357.11	357.11	502.43	6	Generated	0.00
Precedence	Repair broken	371.66	371.66	524.51	6	Generated	0.00
Precedence	Repair broken	371.66	371.66	524.33	6	Generated	0.00
Precedence	Repair broken	371.66	371.66	528.15	6	Generated	0.00
Precedence	Repair waiting	371.66	371.66	555.72	6	Generated	0.00
Precedence	Repair waiting	371.66	371.66	531.59	6	Generated	0.00
Precedence	Repair waiting	371.66	371.66	527.32	6	Generated	0.00
Plugs	Repair broken	370.34	368.91	483.83	6	Generated	0.39
Plugs	Repair broken	368.91	368.91	532.59	6	Generated	-0.00
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Settings	Heuristic solution	Exact solution	Duration	Turbines	Field	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Plugs	Repair broken	369.67	368.91	515.80	6	Generated	0.21
Plugs	Repair waiting	377.19	368.91	577.66	6	Generated	2.25
Plugs	Repair waiting	368.91	368.91	607.37	6	Generated	-0.00
Plugs	Repair waiting	368.91	368.91	587.53	6	Generated	-0.00
Arrival	Repair broken	370.34	369.04	528.82	6	Generated	0.35
Arrival	Repair broken	369.04	369.04	566.37	6	Generated	0.00
Arrival	Repair broken	371.82	369.04	493.72	6	Generated	0.76
Arrival	Repair waiting	369.11	369.04	575.58	6	Generated	0.02
Arrival	Repair waiting	369.59	369.04	570.63	6	Generated	0.15
Arrival	Repair waiting	369.04	369.04	595.83	6	Generated	0.00
Blank	Repair broken	358.37	358.37	509.07	6	Hornsrev	0.00
Blank	Repair broken	358.37	358.37	508.46	6	Hornsrev	0.00
Blank	Repair broken	358.37	358.37	506.20	6	Hornsrev	0.00
Blank	Repair waiting	358.37	358.37	505.79	6	Hornsrev	0.00
Blank	Repair waiting	358.37	358.37	510.31	6	Hornsrev	0.00
Blank	Repair waiting	358.37	358.37	509.92	6	Hornsrev	0.00
Precedence	Repair broken	368.49	368.49	585.19	6	Hornsrev	0.00
Precedence	Repair broken	368.49	368.49	587.70	6	Hornsrev	0.00
Precedence	Repair broken	368.49	368.49	585.23	6	Hornsrev	0.00
Precedence	Repair waiting	368.49	368.49	593.27	6	Hornsrev	0.00
Precedence	Repair waiting	368.49	368.49	594.91	6	Hornsrev	0.00
Precedence	Repair waiting	368.49	368.49	591.33	6	Hornsrev	0.00
Plugs	Repair broken	375.60	368.12	508.72	6	Hornsrev	2.03
Plugs	Repair broken	368.24	368.12	492.67	6	Hornsrev	0.03
Plugs	Repair broken	368.14	368.12	527.89	6	Hornsrev	0.01
Plugs	Repair waiting	368.15	368.12	581.68	6	Hornsrev	0.01
Plugs	Repair waiting	368.20	368.12	582.30	6	Hornsrev	0.02
Plugs	Repair waiting	368.69	368.12	585.54	6	Hornsrev	0.16
Arrival	Repair broken	376.09	368.37	529.03	6	Hornsrev	2.10
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Type	Settings	Heuristic solution	Exact solution	Duration	Turbines	Field	Gap
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Arrival	Repair broken	369.63	368.37	572.74	6	Hornsrev	0.34
Arrival	Repair broken	375.71	368.37	534.04	6	Hornsrev	1.99
Arrival	Repair waiting	368.37	368.37	635.30	6	Hornsrev	0.00
Arrival	Repair waiting	368.79	368.37	629.20	6	Hornsrev	0.11
Arrival	Repair waiting	368.82	368.37	619.13	6	Hornsrev	0.12

D.2.7. Optimised Schedule with arrival

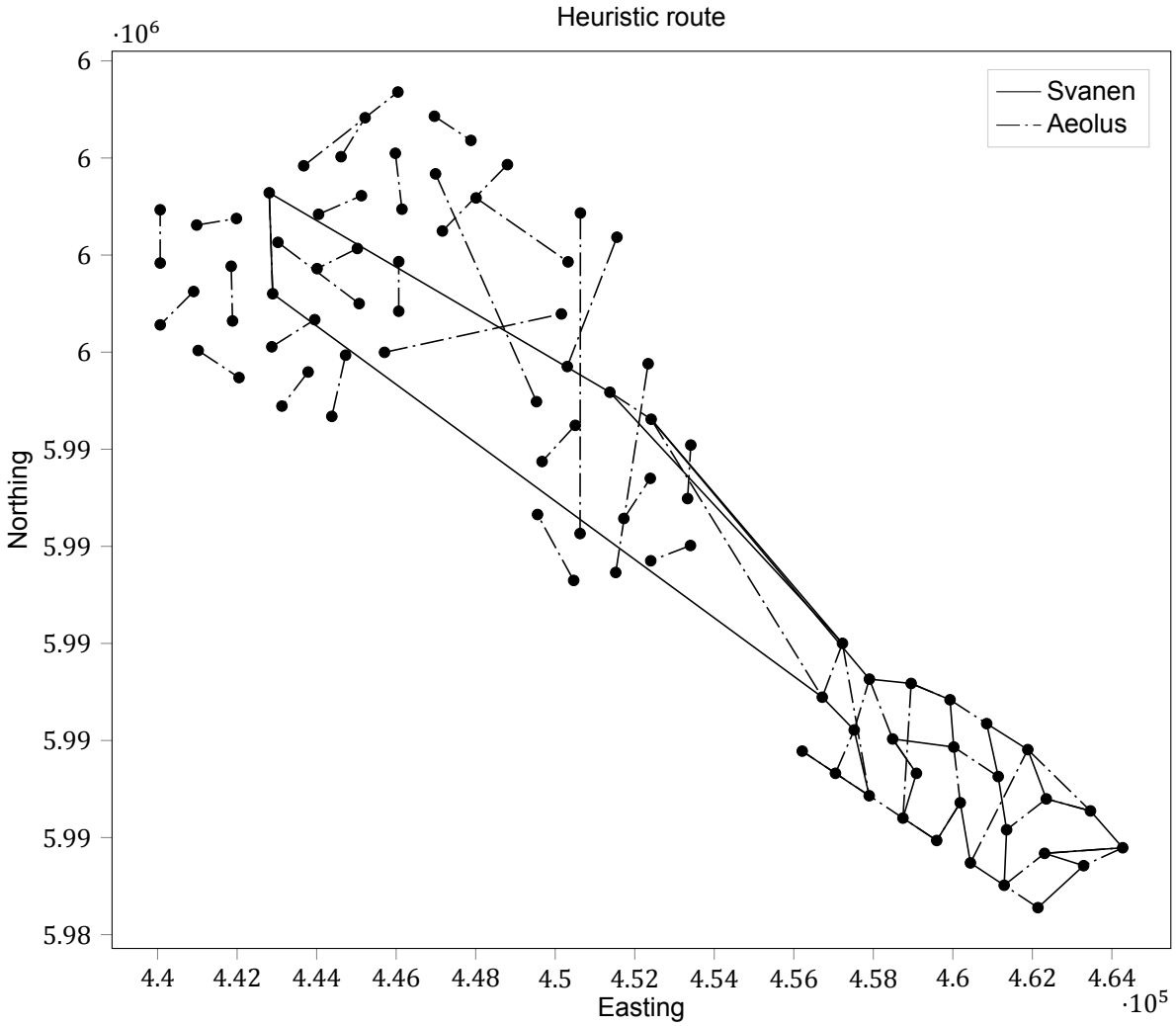


Figure D.4: Optimised walney route with arrival

Object	Location	Time
Aeolus jacked	Harbour	3610
Aeolus jacked	C07	3631
Aeolus jacked	F06	3651
Aeolus jacked	Harbour	3681
Aeolus jacked	D04	3702
Aeolus jacked	D02	3721
Aeolus jacked	Harbour	3751
Aeolus jacked	F01	3773
Aeolus jacked	H01	3793
Aeolus jacked	Harbour	3824
Aeolus jacked	B01	3846
Aeolus jacked	C01	3865
Aeolus jacked	Harbour	3896
Aeolus jacked	C03	3917
Aeolus jacked	B03	3936
Aeolus jacked	Harbour	3966
Aeolus jacked	H04	3988
Aeolus jacked	F05	4007
Aeolus jacked	Harbour	4038
Aeolus jacked	C06	4058
Aeolus jacked	A05	4078
Aeolus jacked	Harbour	4107
Aeolus jacked	B05	4128
Aeolus jacked	A04	4147
Aeolus jacked	Harbour	4176
Aeolus jacked	G04	4198
Aeolus jacked	G05	4217
Aeolus jacked	Harbour	4247
Aeolus jacked	C02	4269
Aeolus jacked	D01	4288
Aeolus jacked	Harbour	4319
Aeolus jacked	F02	4341
Aeolus jacked	G01	4360
Aeolus jacked	Harbour	4391
Aeolus jacked	G02	4413
Aeolus jacked	F04	4432
Aeolus jacked	Harbour	4462
Aeolus jacked	G03	4484
Aeolus jacked	D06	4504
Aeolus jacked	Harbour	4534
Aeolus jacked	E02	4555
Aeolus jacked	F03	4574
Aeolus jacked	Harbour	4605
Aeolus jacked	B04	4626
⋮	⋮	⋮
Aeolus jacked	C05	4645
Aeolus jacked	Harbour	4675
Aeolus jacked	H02	4697
Aeolus jacked	H03	4716
Aeolus jacked	Harbour	4747
Aeolus jacked	D05	4768
Aeolus jacked	E04	4787
Aeolus jacked	Harbour	4817
Aeolus jacked	A02	4838
Aeolus jacked	A03	4857
Aeolus jacked	Harbour	4887
Aeolus jacked	E05	4907
Aeolus jacked	H06	4927
Aeolus jacked	Harbour	4957
Aeolus jacked	A06	4976
Aeolus jacked	B06	4996
Aeolus jacked	Harbour	5024
Aeolus jacked	B07	5044
Aeolus jacked	H05	5064
Aeolus jacked	Harbour	5095
Aeolus jacked	B02	5116
Aeolus jacked	A01	5135
Aeolus jacked	Harbour	5165
Aeolus jacked	C10	5185
Aeolus jacked	B09	5204
Aeolus jacked	Harbour	5232
Aeolus jacked	E03	5254
Aeolus jacked	D03	5273
Aeolus jacked	Harbour	5303
Aeolus jacked	D09	5323
Aeolus jacked	E08	5342
Aeolus jacked	Harbour	5371
Aeolus jacked	B08	5390
Aeolus jacked	F07	5410
Aeolus jacked	Harbour	5439
Aeolus jacked	D07	5460
Aeolus jacked	C08	5479
Aeolus jacked	Harbour	5508
Aeolus jacked	C09	5527
Aeolus jacked	D08	5546
Aeolus jacked	Harbour	5575

Table D.5: Schedule with arrival times for Aeolus jacked

Object	Location	Time
Aeolus floating	Harbour	5815
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Aeolus floating	C17	5835
Aeolus floating	C15	5847
Aeolus floating	B14	5858
Aeolus floating	A11	5870
Aeolus floating	A07	5882
Aeolus floating	Harbour	5902
Aeolus floating	A14	5921
Aeolus floating	A13	5933
Aeolus floating	B16	5944
Aeolus floating	C21	5956
Aeolus floating	B17	5967
Aeolus floating	Harbour	5987
Aeolus floating	A12	6007
Aeolus floating	C18	6019
Aeolus floating	C20	6030
Aeolus floating	C19	6042
Aeolus floating	B15	6053
Aeolus floating	Harbour	6073
Aeolus floating	A10	6093
Aeolus floating	C13	6105
Aeolus floating	C14	6116
Aeolus floating	C16	6128
Aeolus floating	Harbour	6148
Aeolus floating	C04	6170
Aeolus floating	E01	6182
Aeolus floating	Harbour	6205
Aeolus floating	A08	6225
Aeolus floating	B11	6236
Aeolus floating	C12	6248
Aeolus floating	B12	6260
Aeolus floating	B13	6271
Aeolus floating	Harbour	6291
Aeolus floating	E06	6313
Aeolus floating	E07	6324
Aeolus floating	B10	6337
Aeolus floating	C11	6349
Aeolus floating	A09	6361
Aeolus floating	Harbour	6380

Table D.6: Schedule with arrival times for Aeolus floating

Object	Location	Time
Svanen	Harbour	7
Svanen	A14	20
Svanen	B17	30
Svanen	B16	41
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Svanen	C21	52
Svanen	C20	63
Svanen	C19	74
Svanen	C18	84
Svanen	C16	95
Svanen	C17	106
Svanen	B15	117
Svanen	A13	127
Svanen	A12	138
Svanen	B14	152
Svanen	A11	163
Svanen	A10	173
Svanen	B13	184
Svanen	B12	194
Svanen	C15	205
Svanen	C14	216
Svanen	C13	227
Svanen	C12	237
Svanen	E07	254
Svanen	C11	267
Svanen	E06	281
Svanen	E01	295
Svanen	C04	307
Svanen	B10	324
Svanen	B11	335
Svanen	A09	346
Svanen	A08	356
Svanen	A07	367
Svanen	Harbour	390

Table D.7: Schedule with arrival times for Svanen

Object	Location	Time
Plug 0	Harbour	53
Plug 0	C19	74
Plug 0	C13	227
Plug 0	C11	267
Plug 0	C04	307
Plug 0	A08	356
Plug 0	Harbour	384

Table D.8: Schedule with arrival times for plug 0

Object	Location	Time
Plug 1	Harbour	11
Plug 1	B17	30
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Plug 1	C18	84
Plug 1	B12	194
Plug 1	C12	237
Plug 1	E01	295
Plug 1	B11	335
Plug 1	Harbour	362

Table D.9: Schedule with arrival times for plug 1

Object	Location	Time
Plug 2	Harbour	0
Plug 2	A14	20
Plug 2	C20	63
Plug 2	A11	163
Plug 2	C14	216
Plug 2	E07	254
Plug 2	B10	324
Plug 2	A07	367
Plug 2	Harbour	394

Table D.10: Schedule with arrival times for plug 2

Object	Location	Time
Plug 3	Harbour	32
Plug 3	C21	52
Plug 3	C16	95
Plug 3	A12	138
Plug 3	B13	184
Plug 3	E06	281
Plug 3	A09	346
Plug 3	Harbour	373

Table D.11: Schedule with arrival times for plug 3

Object	Location	Time
Plug 4	Harbour	21
Plug 4	B16	41
Plug 4	C17	106
Plug 4	A10	173
Plug 4	Harbour	200

Table D.12: Schedule with arrival times for plug 4

Object	Location	Time
Plug 5	Harbour	108
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Plug 5	A13	127
Plug 5	C15	205
Plug 5	Harbour	233

Table D.13: Schedule with arrival times for plug 5

Object	Location	Time
Plug 6	Harbour	97
Plug 6	B15	117
Plug 6	B14	152
Plug 6	Harbour	179

Table D.14: Schedule with arrival times for plug 6

Object	Location	Time
Plug 7	Harbour	0
Plug 7	A14	20
Plug 7	C17	106
Plug 7	B13	184
Plug 7	E06	281
Plug 7	B10	324
Plug 7	A07	367
Plug 7	Harbour	394

Table D.15: Schedule with arrival times for plug 7

Object	Location	Time
Plug 8	Harbour	32
Plug 8	C21	52
Plug 8	C16	95
Plug 8	A10	173
Plug 8	C14	216
Plug 8	E07	254
Plug 8	A09	346
Plug 8	Harbour	373

Table D.16: Schedule with arrival times for plug 8

Object	Location	Time
Plug 9	Harbour	118
Plug 9	A12	138
Plug 9	Harbour	165

Table D.17: Schedule with arrival times for plug 9

Object	Location	Time
Plug 10	Harbour	0

Table D.18: Schedule with arrival times for plug 10

Object	Location	Time
Plug 11	Harbour	206
Plug 11	C13	227
Plug 11	C11	267
Plug 11	B11	335
Plug 11	Harbour	362

Table D.19: Schedule with arrival times for plug 11

Object	Location	Time
Plug 12	Harbour	64
Plug 12	C18	84
Plug 12	B12	194
Plug 12	A08	356
Plug 12	Harbour	384

Table D.20: Schedule with arrival times for plug 12

Object	Location	Time
Plug 13	Harbour	53
Plug 13	C19	74
Plug 13	A11	163
Plug 13	Harbour	190

Table D.21: Schedule with arrival times for plug 13

Object	Location	Time
Plug 14	Harbour	11
Plug 14	B17	30
Plug 14	Harbour	58

Table D.22: Schedule with arrival times for plug 14

Object	Location	Time
Plug 15	Harbour	108
Plug 15	A13	127
Plug 15	C12	237
Plug 15	E01	295
Plug 15	Harbour	326

Table D.23: Schedule with arrival times for plug 15

Object	Location	Time
Plug 16	Harbour	21
Plug 16	B16	41
Plug 16	B15	117
Plug 16	B14	152
Plug 16	C04	307
Plug 16	Harbour	337

Table D.24: Schedule with arrival times for plug 16

Object	Location	Time
Plug 17	Harbour	43
Plug 17	C20	63
Plug 17	C15	205
Plug 17	Harbour	233

Table D.25: Schedule with arrival times for plug 17

D.2.8. Optimised Schedule without arrival

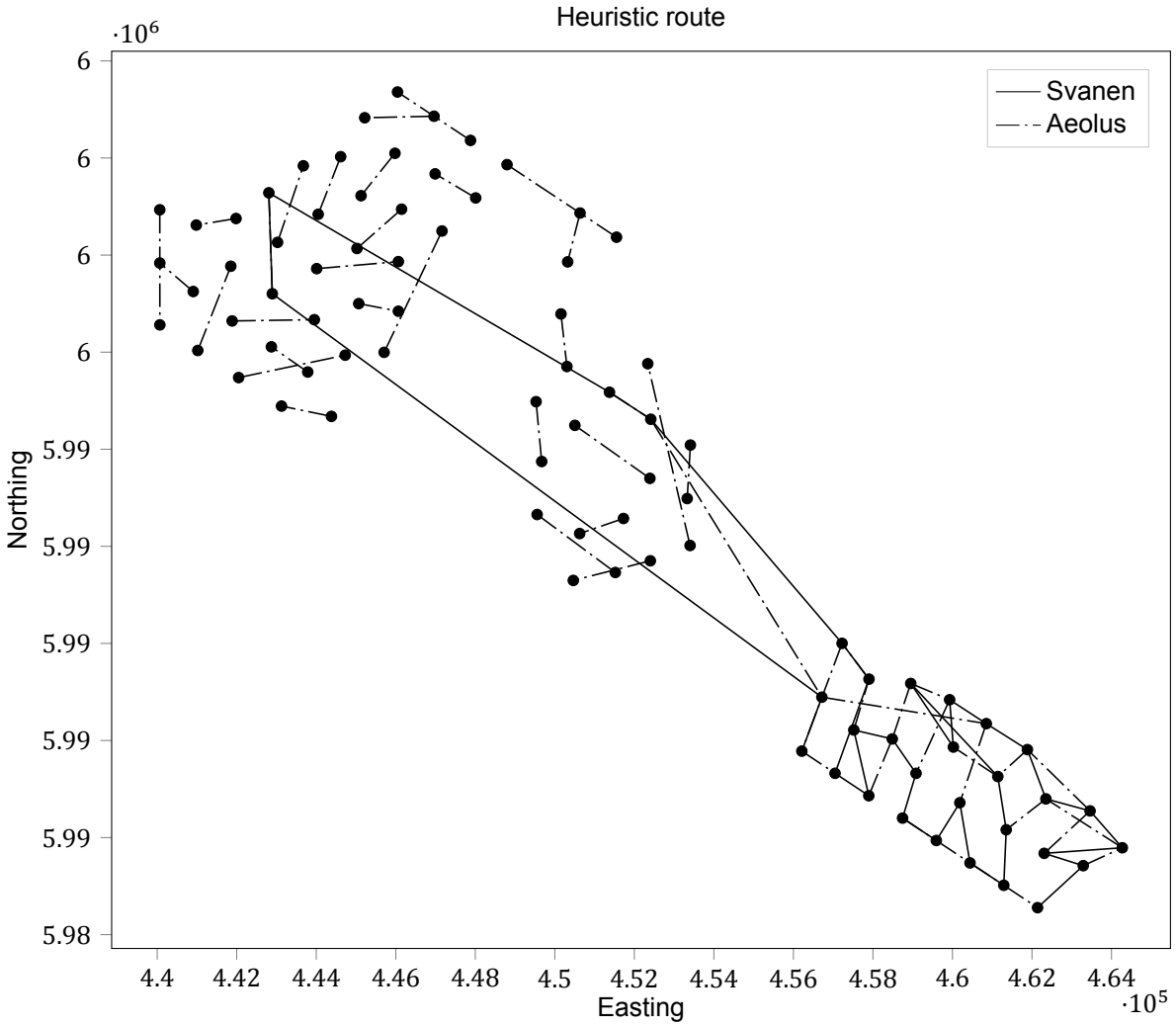


Figure D.5: Optimised walney route without arrival

Object	Location	Time
Aeolus jacked	Harbour	0
Aeolus jacked	B08	19
Aeolus jacked	B06	39
Aeolus jacked	Harbour	67
Aeolus jacked	B02	89
Aeolus jacked	B01	108
Aeolus jacked	Harbour	138
Aeolus jacked	B09	158
Aeolus jacked	A06	177
Aeolus jacked	Harbour	205
Aeolus jacked	D04	226
Aeolus jacked	D05	245
Aeolus jacked	Harbour	275
Aeolus jacked	H06	296
Aeolus jacked	H04	316
Aeolus jacked	Harbour	347
Aeolus jacked	B05	367
Aeolus jacked	B04	387
Aeolus jacked	Harbour	416
Aeolus jacked	D02	438
Aeolus jacked	F01	457
Aeolus jacked	Harbour	488
Aeolus jacked	G04	509
Aeolus jacked	G03	529
Aeolus jacked	Harbour	559
Aeolus jacked	A03	580
Aeolus jacked	C06	600
Aeolus jacked	Harbour	629
Aeolus jacked	F02	651
Aeolus jacked	E02	670
Aeolus jacked	Harbour	701
Aeolus jacked	G02	723
Aeolus jacked	F03	742
Aeolus jacked	Harbour	772
Aeolus jacked	H05	794
Aeolus jacked	G05	813
Aeolus jacked	Harbour	843
Aeolus jacked	C01	865
Aeolus jacked	A01	884
Aeolus jacked	Harbour	915
Aeolus jacked	D03	936
Aeolus jacked	E04	955
Aeolus jacked	Harbour	985
Aeolus jacked	A02	1006
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Aeolus jacked	C03	1026
Aeolus jacked	Harbour	1056
Aeolus jacked	C07	1077
Aeolus jacked	F05	1096
Aeolus jacked	Harbour	1127
Aeolus jacked	A05	1147
Aeolus jacked	A04	1166
Aeolus jacked	Harbour	1196
Aeolus jacked	C09	1215
Aeolus jacked	B07	1234
Aeolus jacked	Harbour	1263
Aeolus jacked	D06	1283
Aeolus jacked	C08	1302
Aeolus jacked	Harbour	1331
Aeolus jacked	H03	1353
Aeolus jacked	H01	1373
Aeolus jacked	Harbour	1404
Aeolus jacked	B03	1425
Aeolus jacked	C05	1444
Aeolus jacked	Harbour	1474
Aeolus jacked	E08	1494
Aeolus jacked	D09	1513
Aeolus jacked	Harbour	1542
Aeolus jacked	D08	1562
Aeolus jacked	D07	1581
Aeolus jacked	Harbour	1610
Aeolus jacked	C10	1630
Aeolus jacked	F07	1650
Aeolus jacked	Harbour	1679
Aeolus jacked	D01	1701
Aeolus jacked	C02	1720
Aeolus jacked	Harbour	1750
Aeolus jacked	E03	1772
Aeolus jacked	F04	1791
Aeolus jacked	Harbour	1821
Aeolus jacked	F06	1842
Aeolus jacked	E05	1861
Aeolus jacked	Harbour	1891
Aeolus jacked	H02	1913
Aeolus jacked	G01	1932
Aeolus jacked	Harbour	1963

Table D.26: Schedule without arrival times for Aeolus jacked

Object	Location	Time
Aeolus floating	Harbour	2203
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Aeolus floating	E01	2227
Aeolus floating	C04	2238
Aeolus floating	Harbour	2261
Aeolus floating	A09	2281
Aeolus floating	B12	2292
Aeolus floating	C13	2304
Aeolus floating	C14	2315
Aeolus floating	B13	2327
Aeolus floating	Harbour	2347
Aeolus floating	B17	2366
Aeolus floating	C21	2378
Aeolus floating	C19	2390
Aeolus floating	B15	2401
Aeolus floating	Harbour	2421
Aeolus floating	B16	2441
Aeolus floating	C20	2452
Aeolus floating	C18	2464
Aeolus floating	C17	2475
Aeolus floating	C15	2487
Aeolus floating	Harbour	2507
Aeolus floating	B11	2527
Aeolus floating	C12	2538
Aeolus floating	C11	2550
Aeolus floating	A07	2562
Aeolus floating	A08	2573
Aeolus floating	Harbour	2593
Aeolus floating	E06	2615
Aeolus floating	E07	2626
Aeolus floating	B10	2640
Aeolus floating	C16	2652
Aeolus floating	B14	2664
Aeolus floating	Harbour	2683
Aeolus floating	A10	2703
Aeolus floating	A11	2714
Aeolus floating	A12	2726
Aeolus floating	A13	2737
Aeolus floating	A14	2749
Aeolus floating	Harbour	2768

Table D.27: Schedule without arrival times for Aeolus floating

Object	Location	Time
Svanen	Harbour	7
Svanen	A14	20
Svanen	B17	30
Svanen	B16	41
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Svanen	C21	52
Svanen	C20	63
Svanen	C19	74
Svanen	C18	84
Svanen	C16	95
Svanen	C14	106
Svanen	C15	116
Svanen	C13	132
Svanen	C17	143
Svanen	B15	154
Svanen	A13	165
Svanen	A12	175
Svanen	B14	186
Svanen	A11	197
Svanen	A10	207
Svanen	B13	218
Svanen	B12	229
Svanen	B11	239
Svanen	A09	250
Svanen	A08	264
Svanen	C12	275
Svanen	C11	286
Svanen	E07	299
Svanen	E06	310
Svanen	E01	324
Svanen	C04	336
Svanen	B10	353
Svanen	A07	364
Svanen	Harbour	387

Table D.28: Schedule without arrival times for Svanen

Object	Location	Time
Plug 0	Harbour	75
Plug 0	C16	95
Plug 0	C13	132
Plug 0	B14	186
Plug 0	B12	229
Plug 0	A08	264
Plug 0	E06	310
Plug 0	B10	353
Plug 0	Harbour	381

Table D.29: Schedule without arrival times for plug 0

Object	Location	Time
Plug 1	Harbour	11
Plug 1	B17	30
Plug 1	C15	116
Plug 1	A12	175
Plug 1	B13	218
Plug 1	C11	286
Plug 1	A07	364
Plug 1	Harbour	391

Table D.30: Schedule without arrival times for plug 1

Object	Location	Time
Plug 2	Harbour	0
Plug 2	A14	20
Plug 2	C20	63
Plug 2	C14	106
Plug 2	B15	154
Plug 2	A11	197
Plug 2	C12	275
Plug 2	E01	324
Plug 2	Harbour	355

Table D.31: Schedule without arrival times for plug 2

Object	Location	Time
Plug 3	Harbour	32
Plug 3	C21	52
Plug 3	A13	165
Plug 3	B11	239
Plug 3	E07	299
Plug 3	Harbour	328

Table D.32: Schedule without arrival times for plug 3

Object	Location	Time
Plug 4	Harbour	21
Plug 4	B16	41
Plug 4	C18	84
Plug 4	C17	143
Plug 4	A10	207
Plug 4	A09	250
Plug 4	C04	336
Plug 4	Harbour	366

Table D.33: Schedule without arrival times for plug 4

Object	Location	Time
Plug 5	Harbour	53
Plug 5	C19	74
Plug 5	Harbour	101

Table D.34: Schedule without arrival times for plug 5

Object	Location	Time
Plug 6	Harbour	0

Table D.35: Schedule without arrival times for plug 6

Object	Location	Time
Plug 7	Harbour	43
Plug 7	C20	63
Plug 7	B15	154
Plug 7	A11	197
Plug 7	B11	239
Plug 7	E01	324
Plug 7	Harbour	355

Table D.36: Schedule without arrival times for plug 7

Object	Location	Time
Plug 8	Harbour	32
Plug 8	C21	52
Plug 8	Harbour	80

Table D.37: Schedule without arrival times for plug 8

Object	Location	Time
Plug 9	Harbour	53
Plug 9	C19	74
Plug 9	Harbour	101

Table D.38: Schedule without arrival times for plug 9

Object	Location	Time
Plug 10	Harbour	0

Table D.39: Schedule without arrival times for plug 10

Object	Location	Time
Plug 11	Harbour	96
Plug 11	C15	116
Plug 11	A12	175
Plug 11	B12	229
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮
Plug 11	E06	310
Plug 11	B10	353
Plug 11	Harbour	381

Table D.40: Schedule without arrival times for plug 11

Object	Location	Time
Plug 12	Harbour	75
Plug 12	C16	95
Plug 12	C13	132
Plug 12	B14	186
Plug 12	C11	286
Plug 12	Harbour	314

Table D.41: Schedule without arrival times for plug 12

Object	Location	Time
Plug 13	Harbour	11
Plug 13	B17	30
Plug 13	C14	106
Plug 13	A13	165
Plug 13	B13	218
Plug 13	A08	264
Plug 13	A07	364
Plug 13	Harbour	391

Table D.42: Schedule without arrival times for plug 13

Object	Location	Time
Plug 14	Harbour	21
Plug 14	B16	41
Plug 14	C18	84
Plug 14	C17	143
Plug 14	A10	207
Plug 14	A09	250
Plug 14	C04	336
Plug 14	Harbour	366

Table D.43: Schedule without arrival times for plug 14

Object	Location	Time
Plug 15	Harbour	0
Plug 15	A14	20
Plug 15	C12	275
Plug 15	Harbour	303
⋮	⋮	⋮

Object	Location	Time
⋮	⋮	⋮

Table D.44: Schedule without arrival times for plug 15

Object	Location	Time
Plug 16	Harbour	277
Plug 16	E07	299
Plug 16	Harbour	328

Table D.45: Schedule without arrival times for plug 16

Object	Location	Time
Plug 17	Harbour	0

Table D.46: Schedule without arrival times for plug 17