Final Report

Pepijn Klop, Sayed Mozafar Shah, Ashwin Sitaram

January 2021

Mentor: Menno van Starrenburg (Systems Navigator)
Client: Mart Jansen (Systems Navigator)
TU Delft Coach: Matthijs Spaan
Bachelor End Project committee: Otto Visser, Thomas Overklift, Huijuan Wang.

Contents

1 Preface

This final report, "Advanced Predictive Planning", is based on the research and development that was conducted during this project. This project forms one of the major graduation requirements of the Bachelor Computer Science & Engineering at Delft University of Technology (TU Delft). This report has been written to exhibit our process, findings, solutions, and recommendations. The project was set up by Systems Navigator as they were interested in providing their clients with a well researched and innovative planning tool. Therefore, we want to thank Mart Jansen and Menno van Starrenburg from Systems Navigator for providing us the opportunity to work on this project and for making time to help us out whenever we had questions.

Also, we want to thank our coach from our university, Matthijs Spaan, for his excellent support and guidance throughout this project; especially during these tough times where we could not meet and discuss issues in person. Not only did he help ensure that we met the academic requirements set by TU Delft, but he also provided us with great insights regarding the practical part of our project.

The most exhilarating part of this project was that we were free to create a product as we pleased, meeting the requirements of the client of course. This gave us an exceptional chance to combine our theoretical knowledge gained throughout our studies and the practical part that is involved in this field. For that, we want to thank the course coordinators that helped organise such a marvellous course: Otto Visser, Thomas Overklift, and Huijuan Wang.

2 Summary

This project has been divided into five phases. The first two weeks formed the research phase. During this phase, research has been conducted into techniques used for scheduling visits. How those techniques work and why the execution of those techniques would lead to an "optimal" schedule were the key points of evaluation. The goal of this research phase was to determine which scheduling techniques could be useful for Systems Navigator's problem.

Afterwards, an investigation phase was performed to come up with an algorithm that could generate a schedule, optimising some Key Performance Indicator (KPI) rating. Furthermore, some of the client's customers guarantee an in order of Estimated Time of Arrival (ETA) processing service or a user-defined priority service, so the algorithm should also be able to generate a schedule that would adhere to this guarantee. At the end of the investigation phase, it was clear that finding an optimal solution to the client's problem efficiently was not tangible because of the complexity of the problem. Reasons for that complexity include *scheduling with required jobs*, *discrete interval scheduling* with dynamic starting times and a fixed number of machines, and more. Furthermore, invalidation due to rules also introduces complexity.

During the following two phases, the implementation phase and modifying phase respectively, the chosen algorithms were implemented and tested. As a starting point a brute force algorithm was written that considered all the permutations and possible allocations of the selected visits. Followed up by the implementation of the heuristic-based algorithms for each KPI.

The departure time KPI is defined as the number of hours the visit is finished after its ETA. The demurrage cost KPI is defined as the delayed number of hours multiplied by the demurrage rate of the visit. For both, the departure time KPI and the demurrage cost KPI, the selected visits are first sorted. The berth time KPI is defined as the duration of the visit on a location, the duration of a visit can vary from location to location. For the berth time, the order of the visits does not matter, as the placement of one visit does not affect the rating of the berth time for another visit on the same location.

The solutions for each of the KPIs is combined to form a schedule optimising all KPIs. To combine them, a new rating has been defined as the sum of each of the KPI ratings of a visit divided by a pre-defined ratio value for the corresponding KPI. This ratio value indicates the importance of a KPI relative to the other KPIs.

Finally, the documentation phase, in which the research, process, findings, and recommendations have been documented. The most important recommendation for Systems Navigator and perhaps other researchers is to perform further research with regards to scheduling techniques to determine whether an exact approach exists that can find an optimal schedule efficiently.

3 Introduction

Systems Navigator is developing Dropboard, which is "a next step in interactive and collaborative online planning systems" [5]. Dropboard's clients are ports and liquid bulk terminals all over the world. These clients use Dropboard to visualise the existing operational schedule, as well as to plan in ships that will be arriving in the future. The following subsections will explain what Dropboard looks like and how it is currently being used.

3.1 Dropboard's schedule page

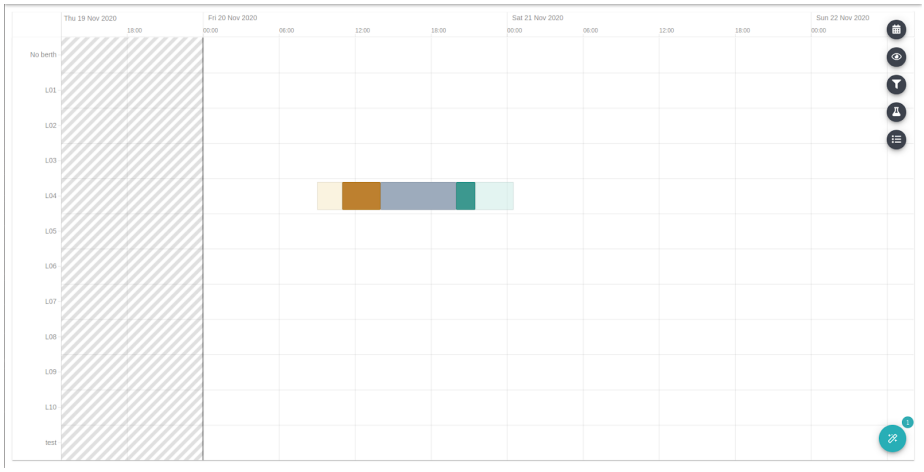

Figure 1: The schedule page

Figure 1 is a screenshot of Dropboard's schedule page, containing a chart. On the horizontal axis of this chart, the time is shown. By clicking and dragging the mouse left or right, a user can move backwards or forwards in time respectively. On the vertical axis, all 'locations' are shown. Locations are explained further in section 3.1.1. In the current schedule, we can see one 'visit'. Visits are explained in detail in section 3.1.2.

3.1.1 What is a Location?

A location is a place where a ship can be moored so that its cargo can be (un)loaded. The technical term for a location is a Jetty or a Berth. Berths and Jetties are very similar, but Berths tend to be bigger and thus are typically used by larger ships. In the screenshot above, we can see that there are 12 different locations. The first one is called 'No berth', followed by 11 other locations. The 'No berth' location is not actually an existing location. It is merely here to

visualise visits that have not been scheduled yet, thus do not have a location to be visualised on yet.

3.1.2 What is a Visit?

A visit is a representation of a ship 'visiting' the client's port or terminal. In Dropboard, a visit is represented by 3 types of tasks:

- The arrival of the ship
- Some n jobs the ship is going to do, where n $\geq$ 1
- The departure of the ship

For a visit, an ETA is given. This depicts the date and time at which a ship will be arriving at the port or terminal. At some point after, or equal to the ETA, the visit's arrival task may be scheduled. During this task, the ship will travel into the port or terminal, to a given location. After the arrival task, a visit has a list of jobs to be scheduled. These jobs are ordered, and cannot be executed in another order. Finally, after all jobs have been scheduled, the departure task of the visit may be scheduled. During this task, the ship will leave the port or terminal and sail away. The order of all of these tasks is set, and can not be changed. The duration of any task may depend on the location it takes place.

In figure 1 a plan with a single visit is given. The line on 20 November 2020 indicates the ETA. The two leftmost squares represent the arrival task. The blue-grey square represents a job. The two rightmost squares represent the departure task. In the y-axis, one can see that all tasks of this visit are scheduled on the location: 'L04'.

3.2 What is a (valid) plan?

Dropboard users can configure so-called 'Rules'. A rule consists of several conditions and a specification of what should happen. With these rules, the user can enforce almost any type of restriction on visits. Some of these restrictions are physical, but they may also be business rules.

For example, certain ships can only go to certain locations, due to their size/ cargo. Other rules can be a bit more complex to evaluate. For example, certain ships may only be at a certain location during a certain time of the day, due to the tidal fluctuations. Another example is, if a ship of size $\geq$ X is at location A, a ship of size $\geq$ Y may not be at location B during the time the former ship is at location A, because that ship is blocking a part of location B.

A valid plan is a plan that meets two requirements: Each of a visit's tasks (Arrival, Jobs and Departure) start at or after its ETA, and all rules regarding that visit are satisfied.

3.3 What is the quality of a plan?

A plan's quality can be measured in a number of ways. For Dropboard's users, there are three KPIs that can be used to quantify and compare plans. The three KPIs are called the departure time KPI, berth time KPI, and demurrage cost KPI. These KPIs are explained in depth in sections 7.1.1, 7.1.2, and 7.1.3 respectively. For each of these KPIs, a lower value is better.

3.4 How are visits scheduled

When a client receives an order for a visit, the visit is entered into Dropboard, with its corresponding information regarding dates and ship details. This visit is then displayed within Dropboard on the 'No berth' as it has not been scheduled yet. To find an allocation for this visit, the user must manually place the visit on a location and press a button to start the scheduling process. After this process, the visit will have been placed optimally on that location, meaning its tasks will start as early as possible, without idle time between them and while meeting all requirements posed by the rules. It is then up to the user to save the visit or to repeat the scheduling process on another location.

4 Problem Definition

The method of scheduling visits as described in section 3.4 is far from ideal. A user can only plan one visit at a time. This is especially bad considering that the allocation of one visit can influence the allocation of all other visits due to specific rules. An example of such a rule was given in section 3.2. Another downside of this approach is that a user must try each location individually, and somehow figure out the 'best' allocation by memorising the allocation on each location.

4.1 Project Description

The goal of this project is to create an algorithm that automatically generates schedules containing multiple visits. The user selects which visits to include in the algorithm, and the algorithm is to find multiple valid plans. A plan can be rated on a number of KPIs. These plans should be presented to the user such that the user can see the rating of each plan (for every KPI) and compare them, as well as persist a plan into the operational plan. A list containing all of the requirements is given in section 4.3.

4.2 Deliverable

The product will be built directly into Dropboard, using any existing code where possible. The product will be used in production upon completion. The product consists of mainly three parts.

Firstly, the user should be allowed to select multiple visits in Dropboard that shall be included in a plan generation. Only the selected visits will be scheduled by the algorithm.

Then, at least three plans will be generated, where each plan is prevalent in a different KPI. Additionally, other plans that are good in multiple KPIs may be presented to the user.

Some of Dropboard's clients schedule their visits according to some business rules. To facilitate this, our algorithm should present two additional plans:

- An ETA sorted plan
- A user-defined priority-based plan

For these plans, the visits should be scheduled in order of their ETA or a user-defined priority. A user-defined priority is represented as a number, linked to a visit. Visits with a higher priority are scheduled before visits with a lower priority.

Finally, the generated plans are presented to the user and a ranking with regard to the KPIs will be provided. The user may select a plan to view it on the

schedule. After a user has selected the best option, the plan should be applied to Dropboard's operational plan.

4.3 Requirements

This section mentions all of the requirements for this project, as agreed upon by Systems Navigator. These requirements are sorted based on their importance. The most important requirements are the must haves, and the least important ones are the won't haves.

4.3.1 Must haves

The following requirements are the minimal requirements that the solution should adhere to. These requirements ensure that the program at least performs the necessary tasks.

- The clients must be able to select any number of visits for which they want to be given a schedule optimising according to various KPIs. These KPIs must include:
  - Departure time (the departure is defined as the n.o. hours the ship leaves a port after the indicated ETA)
  - Demurrage cost (penalty for finishing later than an indicated range of time)
  - Berth time (sum of the durations of the tasks of a visit)
- A user friendly UI must be provided in which the clients must be able to:
  - See the visits
  - Select the visits for which they want to receive a schedule
  - See the resulting schedules based on different KPIs, the clients must be able to click on a specific KPI to see the schedule that optimises accordingly
  - For each KPI a ranking of the suggested schedules must be provided
- The clients must be provided multiple suggested schedules according to each KPI.
- The clients must be provided suggested schedules that are "optimal" according to a ratio between different KPIs.
- The clients must be able to select a suggested schedule as the schedule they want to use and the visits should be planned accordingly.

4.3.2 Should haves

- The clients should be able to add visits to a calculated schedule
  - They should be able to choose whether an entirely new schedule should be calculated, or the current schedule should be edited such that the added visit(s) are planned within the schedule as optimally as possible
- The clients should be able to select (a) specific KPI(s) for which they want to receive suggested schedules, by opting out the KPI(s) they do not want. (By default every KPI is selected)
- The product could support optimisation according to multiple locations for every job in addition to the KPIs. Meaning every possible location for every job is considered.

4.3.3 Could haves

- The clients could be able to select the locations that have to be considered in the algorithm
- A list of all locations could be provided to the clients, in which they can select the locations they want to be considered for the suggested schedules. (By default all locations are selected)
- The clients could be able to order the KPIs according to their priority, such that the suggested schedules will be ranked based on their indicated priorities.

4.3.4 Won't haves

These are the requirements that will not be implemented during this development phase. The absence of these requirements will not influence the functionality of the product. These requirements could be considered as features to be added in the future.

- The product won't have the possibility to give a specific job a priority, so no additional parameters beside the already defined KPIs.
- The product won't have automated scheduling, the user has to select the visit that will be included in the schedule.
- The product won't have an algorithm that always returns the optimal schedule.
- The product won't have a feature in which multiple ships can be processed on the same location.

5 Research

The purpose of performing this research is to gain insight into the techniques used for scheduling tasks, jobs, processes, etc., how these techniques work, and why these are the "optimal" solution for a given scheduling problem. Also, doing research can perchance be beneficial to avoiding mistakes in the software solution that has to be built by noticing mistakes made by others and/ or learning from any creative methods applied by others.

In a scheduling or planning system, precise predictions are fundamental in ensuring that the clients can optimally use their resources given their scarce amount of time. Looking at a port as an example, to be able to make precise predictions it is necessary to make accurate estimations regarding the arrival time of a ship, the duration of (un)loading a ship, and the time it takes for a ship to enter and leave the dock. Furthermore, consideration of various other constraints such as the type of ship (some ships are only allowed to dock at specific stations), the size of the ship, the materials it is carrying, etc. are needed to punctiliously predict a time span at which location is unavailable.

However, each client's needs differ and consequentially their definition of an optimal schedule differs. One client may want to minimise the waiting time (of a ship in the port example), another may want to minimise the number of resources needed to process all visits on a given day. Thus, the planning system should be able to find solutions for numerous types of "optimal" schedules. This is another limitation of the current version of Dropboard, clients cannot choose what variables are taken into consideration when a planning suggestion for a single visit is given.

5.1 Research questions

Throughout this section the following question will be answered: how to generate multiple multi-visit valid plans and rank them based on certain KPIs? To answer this question, the following sub-questions need to be answered first:

1. How to generate a valid plan?
2. How to generate a good valid plan with regard to a certain KPI?
3. How to generate multiple plans that differ from each other and select the best options?

5.2 Research aspects

The section Scheduling Techniques (section 5.3) explains how these techniques work, and why these are the "optimal" solution for a given scheduling problem. In addition to researching scheduling techniques, it is also fundamental to research the current functionality of Dropboard and to use the existing code base in order to solve problems with algorithms that are already in Dropboard.

As described in the project description, Systems Navigator is looking for an advanced predictive planning solution for Dropboard. The knowledge gained by performing this research will be used to determine what techniques are most helpful to improve and build the functionality to generate plans in Dropboard.

5.3 Scheduling techniques

To optimise a schedule according to a specified notion of optimality, such as a schedule that ensures the least number of resources are used to execute all tasks or a schedule that ensures the maximum lateness of a set of visits is minimised, etc., certain programming techniques of various algorithmic paradigms are used to determine a schedule that adheres to the corresponding notion of optimality. That programming technique is referred to as a *scheduling technique* or scheduling algorithm. The algorithmic techniques that are used or necessary to find a solution as fast as possible is determined by the type of scheduling problem that has to be solved.

5.4 Interval Scheduling

Knowing the starting times and or the duration of a task (job) is characteristic in interval scheduling. When discussing scheduling problems, in which there are resources (machines) and tasks that have to be carried out, basic scheduling problems often include (i) finding a specific ordering of tasks such that the number of machines needed to execute all the tasks is minimised (Interval Partitioning), (ii) finding a set that contains the most compatible jobs (i.e. the intervals of the respective jobs do not overlap, where the interval is defined as the period from the start time up to the finish time), (iii) scheduling to minimise lateness and possibly quite a few more that are not mentioned. Often in theoretical solutions to these problems, a major assumption is made about the set of jobs and that is that the scheduler is aware of all the jobs, whereas in practice it might happen that new requests come in after the scheduler has made a schedule [6]. A lot of research has already been performed with regards to finding solutions to various versions of those problems. Some of the techniques that resulted from that research will now be mentioned. Further analysis of those techniques will be documented in section 5.9.

5.4.1 Greedy

Suppose a set of jobs $j = \{1, ..., n\}$ is given, where each job j has a specified starting time and finishing time, s_j and f_j , respectively. The starting times are always smaller than the corresponding finishing times. Another variant might be where the duration d_j of each job is given instead of the finish time, the finish time is then simply determined by $f_j = s_j + d_j$. If the first variant were given the duration could be equally easy to determine. This means an interval is defined for each job of the form $[\, s_j,\ f_j)$ in which the job is (supposed to be) processed. Machines are continuously available as long as they are not processing a job and

machines can process only a single job at the same time. Another assumption is that jobs are only assigned to one machine. Assume that s_j, d_j, and f_j are non-negative numbers (this assumption can be made without loss of generality of the algorithms) [7]. However, for most of the greedy solutions to such interval scheduling problems certain assumptions have to be made about the jobs. One of these assumptions is that each job has the same cost, value, or weight. Another assumption or constraint, which also holds for other programming paradigms, is that during the specified interval of a given job j the job cannot be interrupted. These assumptions are necessary for the algorithms to be significantly faster than a simplistic brute force approach.

Greedy algorithms often require sorting the set of jobs on some property, by doing so the solution, to interval scheduling problems such as i, ii, and iii (see 5.4) with the constraints mentioned before, can be found more efficiently in comparison to other algorithms.

5.4.2 Interval Graph

Interval graphs can also be used to solve interval scheduling problems. An interval graph is defined as an undirected graph in which each vertex represents an interval, as defined in the Greedy section (section 5.4.1), and two vertices are connected by an edge (i.e. they are adjacent) if and only if the corresponding intervals overlap at least partially [4]. Restricting it to interval scheduling, this means that it is an undirected graph $G = (V, E)$, where V is the set of vertices representing jobs $j = 1, ..., n$ and an edge between vertices V_s and V_t exists if and only if $V_s \cap V_t \neq \emptyset$. The operation to be performed on the interval graph obviously depends on the problem that needs to be solved. For example, in problem i (see 5.4) the chromatic number (minimum number of colours) can be derived by applying a minimum vertex colouring (*optimal colouring*) of the graph. The chromatic number now represents the minimum number of resources needed to perform all jobs. The optimal colouring problem is equivalent to the *Vertex Colouring Problem*, which is defined as: "Given an undirected graph, the Vertex Coloring Problem (VCP) requires to assign a color to each vertex in such a way that colors on adjacent vertices are different and the number of colors used is minimized." [8]

The Vertex Cover Problem is an NP-hard problem. However, according to Malaguti et al. [8], an exact approach to finding a solution for the Vertex Cover Problem exists. Other sub- optimal approaches, but significantly faster, such as a greedy approach towards finding the chromatic number are also known. [2]

5.5 Weighted interval scheduling

In weighted interval scheduling there is, in addition to the starting time and duration, also a weight associated with every job. The jobs are defined as a set $A = \{j_1, j_2, ..., j_n\}$ of n jobs where every job j contains a start time s_j,

a finish time $f_j > s_j$ and a weight $w_j > 0$. The objective is to maximise or minimise the sum of all weights of the requested plan. The output is a set of non-overlapping intervals that maximise or minimise the sum of all weights w_j. In the case of Systems Navigator this is, for example, the demurrage costs. Due to this new weight parameter, it is not possible to find the optimal solution with a greedy approach [10]. Unweighted interval scheduling problems can be solved by weighted interval scheduling algorithms by giving every job the same weight.

5.5.1 Dynamic Programming

Dynamic programming (DP) is a way to solve a complex problem by dividing the problem in sub-problems that can be solved. This is the reason dynamic programming is often used to solve weighted interval scheduling problems. A problem can only be solved with dynamic programming if it satisfies the optimal substructure and overlapping sub-problems properties. The optimal substructure property states that to solve the global problem optimally, every sub-problem should also be solved optimally [10]. The overlapping sub-problems property states that sub-problems overlap instead of having only distinct sub-problems. To solve a DP problem, the problems need to be divided into smaller sub-problems. This is done by a recursive formulation with an OPT function that defines the value of the optimal solution. [3] Usually, a Dynamic Programming algorithm for weighted interval scheduling consists of the following elements [9]:

- Define a formula for the optimal solution to the problem.
- Define a recursive structure such that the combination of solutions of the sub-problems will give you the root solution.
- Define a memoisation array.
- Fill the memoisation array with solutions to all sub-problems.
- Create an iterative bottom-up algorithm that uses the computed solutions in the array.

5.5.2 Discrete Number of Machines

As mentioned before, interval graphs can be used to solve interval scheduling problems, such as the *Interval Partitioning Problem* (see 5.4). Finding an optimal colouring of the interval graph is equivalent to finding a solution to the Interval Partitioning Problem. Now suppose that there are a discrete number of machines m, where each machine i has an interval $[\, a_i, b_i)$ during which it is available. Outside of this interval, the machine is no longer available contrary to the continuously available machines in the simpler variant of the Interval Partitioning Problem. It can be shown that the problem, which arises from having a discreet number of machines that are discretely available, is polynomially equivalent to the *circular arc colouring problem* [7]. A Dynamic Programming (DP)

approach and a breadth-first approach are suggested by Kolen et al. [7], since a general DP algorithm has been described, only a description of the breadth-first approach towards solving this kind of problem will be given.

The suggested breadth-first algorithm [7] assumes that at each moment t the number of active machines (or locations in this case) equals the number of jobs containing the time t in their chosen interval. So-called 'dummy jobs' are added to ensure that. First, the starting and finishing times, s_j and f_j respectively, of the n jobs are sorted. Then, they are renamed $u_1, u_2, ..., u_k$ such that $u_1 < u_2 < ... < u_k$ ($k \leq 2n$). The 'dummy jobs' will be added whenever the number of jobs is less than the number of available machines in some interval $[\, u_i, u_{i+1})$. The 'dummy jobs' j will be assigned the start time $s_j = u_i$ and the finish time $f_j = u_{i+1}$. Partial schedules are used in this algorithm. They are defined as: "*A partial schedule up to u_i is a schedule that has assigned all jobs that start before u_i to the machines*" [7]. All feasible partial schedules up to u_i should be enumerated, where a feasible schedule is a schedule in which "*no two jobs overlap and each job can be carried out by the machine it is assigned to*" [7]. Given a partial schedule up to u_i, this breadth-first algorithm will consider all jobs starting at u_i and explore all possible assignments of these jobs to the machines. According to Kolen et al. [7], by detecting duplication of feasible partial schedules, the number of schedules to be tracked each iteration is still bounded, and to decide feasibility only non- equivalent schedules need to be maintained. In a situation where the number of machines is fixed, such as the number of locations, this algorithm will run in linear time over the number of jobs.

5.6 Generating different plans based on a singular KPI

There are two solutions that will be considered to solve this problem. Firstly, it is a possibility to write a different algorithm for each KPI that exists. Each of these algorithms would generate a plan, that optimises the ranking of a given KPI in that plan. A benefit of this approach would be that an algorithm is highly specific, thus can be relatively simple and fast. A different scheduling technique that is best suited for a given KPI could be used in each algorithm. A downside is that this approach is not abstract, thus if later a different KPI was to be considered, a new algorithm would have to be written to facilitate this.

A different approach would be to create one single algorithm that takes a number of arguments representing a KPI as input and generates a plan. A benefit of this approach would be that the algorithm is abstract, thus can be tweaked a bit in order to facilitate different KPIs. A downside is that this algorithm would be considerably more complex and the opportunity to use a different scheduling technique tailored for a singular KPI is no longer present.

5.7 Generating a plan based on multiple KPIs

When generating plans for multiple KPIs, it becomes impossible to compare them to each other because it is not possible to compare the number of hours of berth time and departure time to the costs of the demurrage. If one plan has a higher rating in one KPI, but a lower rating in the other KPIs, it should be possible to determine whether this plan is better or worse.

In order to do so, the ratings for each individual KPI could be divided by some ratio. These divided ratings could then be accumulated to form one absolute rating. This rating can then be compared to the absolute rating of other plans in order to determine the superior plan. It is important to note that these ratios are highly client-specific, and must be customisable.

5.8 Forward- Backward Improvement (FBI)

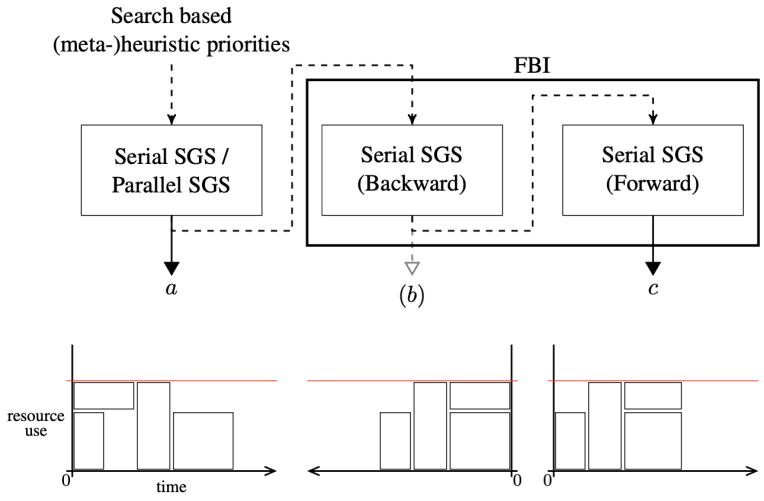

Figure 2: An overview of the FBI algorithm [11]

One of the post scheduling optimisation algorithms that is widely used in Serial and Parallel Schedule Generation Schemes, is the Forward- Backward Improvement or Justification(FBI) algorithm. This algorithm makes two passes over a given schedule. The first pass considers the finish times of the jobs in the schedule in descending order. The last job, according to the initial schedule (schedule a in the figure above), is placed at time $t = 0$, followed up by the second to last job and so on. This will result in a backward schedule relative to the original schedule (schedule b in the figure above). The second pass of the FBI algorithm performs the same operation as the first, but then applied

to the schedule computed in the first pass, this will result in a logical schedule (schedule c in the figure above) following the order of the original schedule [11].

According to Stephen F. Smith [12] a **justification** pass cannot increase the latest finish time, therefore each pass of the FBI algorithm has a chance of improving the original schedule.

5.9 Best technique for Systems Navigator

It is important to choose the scheduling technique that fits the needs of Systems Navigator. Every technique works best on slightly different problems, there is no 'one solution fits all'. Systems Navigator has a complex problem because of the following reasons:

- Scheduling with required jobs (each job needs to be carried out)
- Discrete interval scheduling problem (dynamic starting times)
- Interval scheduling with a fixed number of machines
- Interval scheduling with machine availability (some jobs can only be placed at certain machines)
- Interval scheduling with rules (for example a cargo ship is not allowed during certain time slots)

Unfortunately, there is no efficient algorithm to solve this problem optimally. Because of this, a more advanced greedy algorithm is chosen. This will not guarantee the optimal solution, but will always give a good enough solution. This greedy algorithm is much faster but at the cost of not always giving the optimal solution. The original greedy algorithm is not meant for weighted interval scheduling so we made some adjustments to the algorithm. This works as following:

Listing 1: Pseudo code advanced greedy algorithm

```
Use heuristic to sort visit array

for each visit:
    initialise optimalLocation
    for each location:
        placement = find optimal placement on location
        if placement better than optimalLocation:
            update optimalLocation
    plan visit on optimalLocation
```

First, the array of visits is sorted. The sorting is done using heuristics that translate to one (or possibly more) KPI(s). After the array has been sorted, the visits are scheduled in that order. By evaluating every possible location for a given visit, the best allocation for this visit will be found.

5.9.1 Solution quality

The order in which the visits are scheduled directly determines if the solution will be optimal or not. Thus, by trying multiple permutations, we improve the chance of finding an optimal solution.

A possible approach would be to predict the optimal order as mentioned above and then create a number of random mutations stemming from that order. Another way would be to incorporate likelihoods in the heuristics. If two visits are compared and sorted, a probability could be added to the result of this comparison. If this probability is greater than or equal to a predefined threshold, take the order of those two visits as a 'fact'. However, if the probability is lower than the threshold, create one variant of the sorted array with visit one before visit two, and one variant with visit two before visit one.

5.10 Scheduling a single visit

When scheduling a single visit, all possible locations are evaluated. Existing code in Dropboard is used to do this evaluation. After all locations are evaluated the results are compared and the best location to schedule the visit on is chosen. The "best" allocation depends on the KPI and is therefore evaluated differently for different KPIs.

5.11 Research conclusion

Throughout this report the following research questions have been discussed: How to generate multiple multi-visit valid plans and rank them based on certain KPIs? To answer this question, we have looked for answers to the following sub-questions:

1. How to generate a valid plan?
2. How to generate a good valid plan with regard to a certain KPI?
3. How to generate multiple plans that differ from each other and select the best options?

To answer these questions we have looked at various scheduling techniques that are currently being used (see section 5.3), so we could determine what techniques would be useful for our defined use case. Since we have to generate multiple plans we also looked at ways to generate these plans. As it is up to the scheduler to decide what KPI should be prioritised, multiple plans need to be generated according to various KPIs. Each of these plans should be prevalent according to their respective KPI. We mentioned two approaches for generating multiple plans. Firstly, writing an algorithm for each KPI, making it easy to implement, but also very specific. Secondly, it is possible to create a single algorithm that takes several arguments representing a KPI as input and generates a plan accordingly.

6 Software design

Designing a high quality and high performing algorithm was very important for this project. The algorithm has to be production ready and meet the quality standards of Systems Navigator. In this section the software architecture and design choices of the algorithm are explained.

6.1 Software architecture

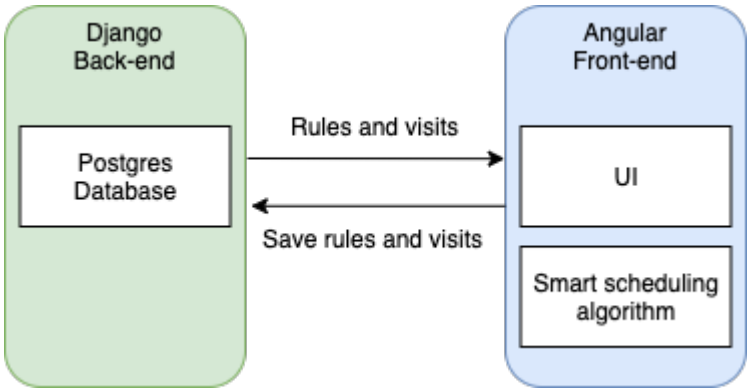

Figure 3: An overview of the software architecture

Dropboard uses the Angular framework for the front-end and Django for the back-end. Angular is an open-source web application framework based on TypeScript. Django is a web-framework used for rapid development written in Python. The Django back-end is connected with a Postgres database, which contains the client's data.

Dropboard does its validation of rules in the front-end because after every change in the schedule all visits have to be validated. Systems Navigator made this decision because it is faster to recalculate in the front-end than doing a request to the back-end every time. Our algorithm needs this validation to create valid plans which is why our code runs in the front-end. We did not use any other tools than the features already present in Dropboard. Unfortunately, the validation of visits takes quite some time. This is further explained in section 8.2.2.

All of our code is written in the front end. The majority of it is written as plain typescript functions. In order to create the UI, we have created one angular component with a corresponding HTML and scss file. In addition, we have created one angular service to exchange data between our UI component and the rest of Dropboard.

6.2 Algorithm design

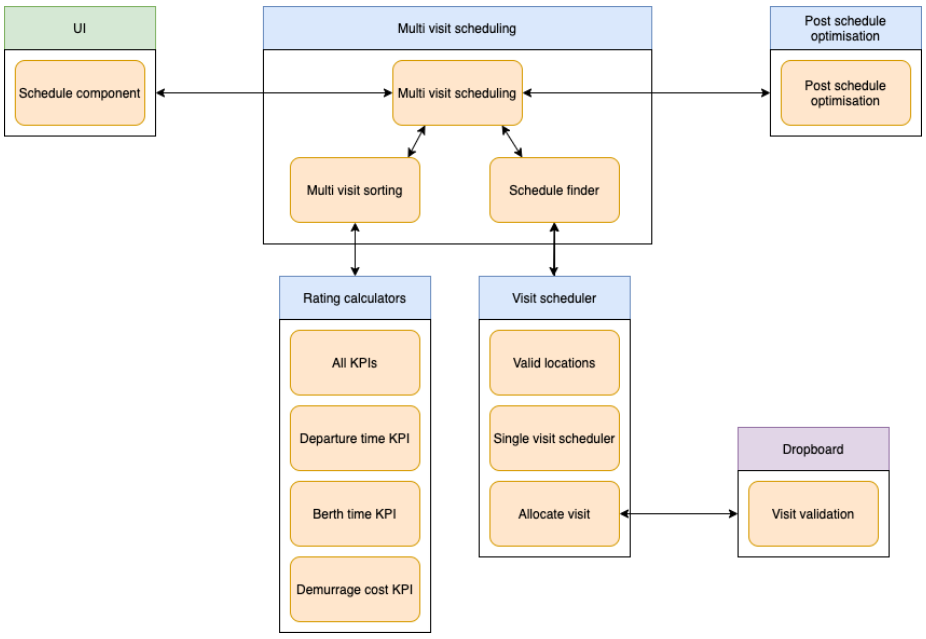

Figure 4: An overview of the communication between files/classes

When multiple visit are selected for scheduling in the UI they are passed to the multi-visit scheduling file. This file coordinates all phases of the algorithm. When the algorithm is done it will send all best plans back to the UI.

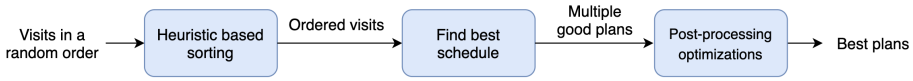

Figure 5: An overview of the algorithm pipeline

The smart scheduling algorithm consists of 3 phases as shown in the figure above. First, in the heuristic-based sorting we sort the visits to create permutations that are more likely to give a good plan. After this, our algorithm finds the best plans of these permutations. These plans are then run through the post-processing optimisation phase to find an improved plan by looking at all delayed visits. All details of our algorithm can be found in section 7.

6.3 UI software design

Please note that the referenced figures are proprietary and are not included in this report.

To use our algorithm, the user needs to select which visits are eligible to be scheduled. The user does so by ctrl + clicking them in the chart, or selecting them in the so-called 'visit list'. Once all desired visits have been selected, the user may press the magic wand to start the scheduling process. This is demonstrated in figure 12.

Once all plans have been generated, the best options are presented to the user in a panel. For each plan, the cumulative rating in each KPI is shown. Next to the rating, the delta to the operational rating is shown, where green numbers indicate an improvement, and red numbers indicate a deterioration. If a plan is clicked, the plan will 'open', and all visits will be visible with their respective rating for each KPI. A visit can be clicked to reveal it, meaning the chart will pan to the start and end of that visit and highlight the tasks of that visit. This is illustrated in figure 14.

An opened plan can be saved using the save button in the panel, or the user can decide to cancel the process using the cancel button in the panel. If the plan is saved, all visits are saved on the location and time mandated by the plan and the panel is closed. If the process is cancelled, all visits are reverted to their state prior to smart scheduling them and the panel is closed.

7 The implementations

The client's request consisted of multiple problems that had to be solved. The main problem that was discussed in section 4 is split up into smaller sub-problems such that solving them becomes easier and the task at hand becomes clearer. The main sub-problems are finding a schedule for a single KPI (1) and finding a schedule for all KPIs (2). Sub- problem 1 consists of three parts as there are three KPIs that should be considered. For each of these three KPIs, the departure time, the berth time, and the demurrage cost, a schedule is generated. The solutions to the three parts of sub-problem 1 are combined to find a solution for sub-problem 2. First, the solution to finding schedules for a single KPI will be explained including a description of the corresponding KPI. Afterwards, the solution to creating a schedule or plan for multiple KPIs is discussed. Followed up by, the effects of post-processing on schedules and its implementation in this product. Finally, the algorithm for finding schedules for sub-problem 1 and sub-problem 2 will be discussed.

7.1 Heuristic-based sort

Various different parameters are of importance among the three KPIs. For example, the value of the departure time KPI for a visit depends on its duration and possible starting time, where the duration also depends on the chosen location. Whereas, for the demurrage KPI other variables such as the laycan start, the laytime of the visit, the demurrage rate, etc. is of greater importance (these variables are explained in section 7.1.3). The various sorting algorithms that have been used to put the selected visits in a useful order according to respective KPI will be explained in this section. Also, how the value or rating of each KPI for a visit is calculated will be explained.

7.1.1 Departure time KPI

The departure time KPI is relatively simple to calculate. As the name indicates, the value or rating of this KPI represents how many hours after the ETA the visit departs. However, the departure time cannot be inferred by the ETA and the duration of the visit directly, as the duration of the visit depends on it's location and when the visit can actually start, which does not always coincide with the ETA.

Figure 6 helps visualise how the departure time KPI is calculated. It shows a single visit at three locations. The value or rating of the departure time KPI is defined as finish time - ETA. The duration of the visit is determined by the locations: location 1, location 2, and location 3. As can be seen in figure 6, at location 1 the duration of the visit is shorter than on location 2, i.e. the departure time has a lower value when placed at location 1. Location 3 is unavailable up to time 'visit start'. The visit could have started immediately after the red block denoted by 'Unavailable', but to avoid messiness the visit is

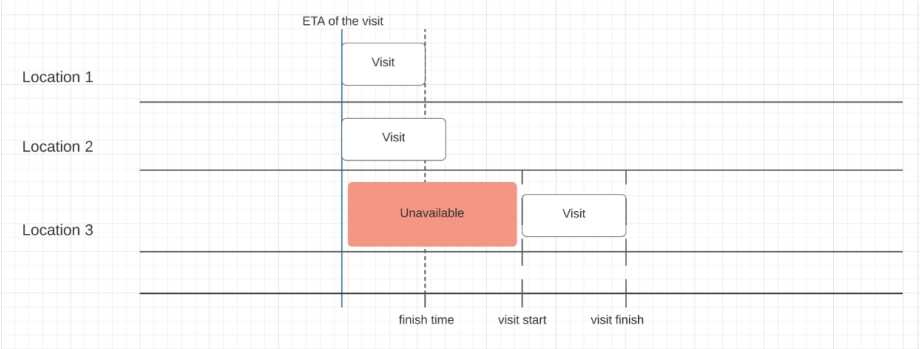

Figure 6: An example of calculating the departure time KPI

planned somewhat later in the graph. The duration of this visit at location 2 is equal to the duration on location 3, however, the rating of the departure time KPI is not the same as on location 2, because the difference between the finish time and the ETA is greater at location 3. Thus, the deciding factor for the rating of this KPI is not only its duration, but also when it starts. The goal is to create a schedule that minimises the value of the departure time KPI.

Algorithm 1: Comparator function used to sort an array of visits

```
function compareDeparture(a: Visit, b: Visit)

  if a.end.getTime() !== b.end.getTime() then
    return a.end.getTime() - b.end.getTime();
  end if
  if a.validLocations.length !== b.validLocations.length then
    return a.validLocations.length - b.validLocations.length;
  end if
  return b.duration - a.duration;
```

Algorithm 1 is used to sort an array of visits to form a preliminary order in which the visits should be considered to create a valid schedule or plan. The algorithm sorts the visits based on end time or finish time of a visit in ascending order. When the end time of two visits is the same, the algorithm considers the number of locations that the visit can be placed. The visit with the least number of valid locations has to be planned first, because that visit is not as 'flexible' as the other visit since it has fewer options. As a final step, whenever two visits also happen to have the same number of valid locations the algorithm sorts the visits based on the duration in descending order. This is done in descending order because the visit with the longer duration has a bigger impact on the departure time rating. In the example of figure 6, the visit has the best departure time rating on Location 1.

7.1.2 Berth time KPI

The berth time KPI is the simplest KPI to calculate. It is somewhat easier than the departure time KPI as the placement of one visit does not affect the berth time value of another visit. When calculating the berth time some similarities with the departure time KPI can be seen. These similarities and calculation of the berth time will be explained with the aid of figure 6.

The berth time is defined as the time it takes to process a visit. As explained in subsection 7.1.1 the duration of a visit can vary from location to location. For example, the berth time of the visit in figure 6 at location 1 is smaller than the berth time at location 2 or 3. This duration is calculated as $finish\ time - start\ time$, which is almost the same as the departure time KPI. What makes the berth time KPI easier to calculate is that the placement of a visit on a given berth does not affect the berth time of another visit on the same berth or location, it depends solely on the duration and not on the start time of the visit. The goal for a plan that 'optimises' according to this KPI is to minimise the value or rating of the sum of berth times on each berth.

As the berth time of each visit is independent of the start time of the visit the sorting of the visits will not impact the value or rating of the berth time KPI. So, the order of the visits does not matter when creating a plan that minimises the sum of berth times. In the example of figure 6, the visit has the best berth time rating on Location 1.

7.1.3 Demurrage cost KPI

The demurrage cost KPI is the most difficult to calculate as multiple variables play a role in the calculation of the demurrage cost. These variables are:

- Laycan start: A pre-defined time at which the window within which the visit has to be handled can start.
- Laytime: A pre-defined duration of time, the actual start + laytime indicates the time after which the visit should start departing.
- Demurrage rate: The fine per unit of time (e.g.: hour) that should be paid if a visit is not handled within the determined time window.
- ETA: Estimated time of arrival of a visit.
- ETB: Estimated time at which the pre-jobs task of a visit starts.
- ETD: Estimated time at which the post-jobs task of a visit ends.

The first three cases in figure 7 are cases in which the demurrage cost is zero. These cases show the various situations that can occur when calculating the

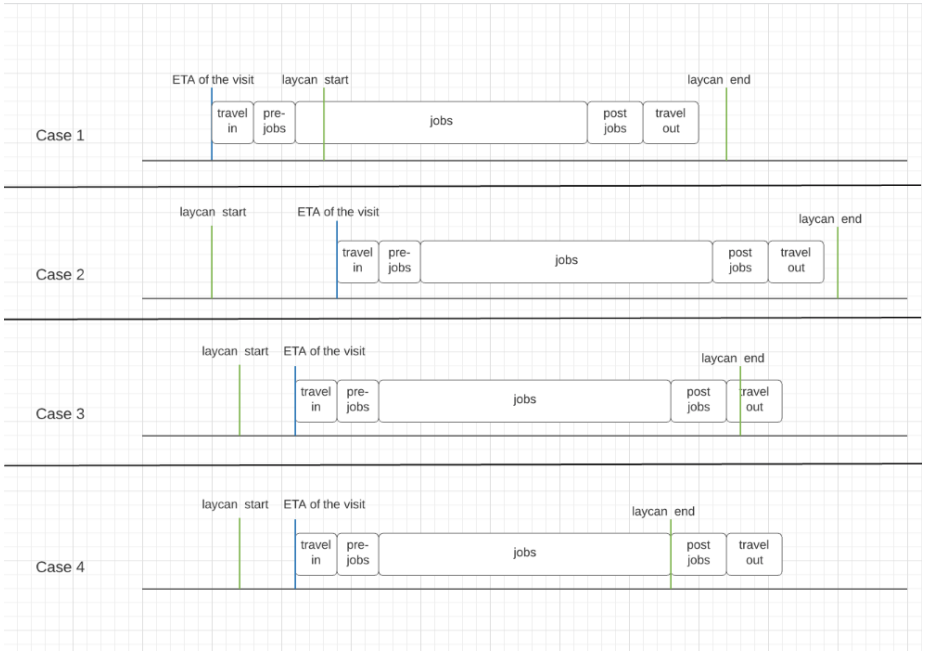

Figure 7: An example of calculating the demurrage cost KPI

demurrage cost. In case 1, the ETA of the visit takes place before the defined laycan start, in such a situation either the laycan start or ETB is chosen as the start time of the 'demurrage window', where the demurrage window is the window from the *actual start* up to *actual start* + *laytime*. The start time is chosen by taking the minimum value between laycan start and ETB ($Min(laycan\ start, ETB)$). However, if the ETA takes place after the defined laycan start, which happens in case 2, then the ETA will be taken as the start of the 'demurrage window'. In figure 7 laycan end indicates *demurrage window start* + *laytime*. If the visit has not been handled up to this point in time, then a fine, which is referred to as the demurrage cost, has to be paid. The visit is timely handled whenever $ETD \leq laycan\ end$. Thus, the complete formula for calculating the demurrage cost for a visit can be written as:

$$\begin{aligned} h &= ETD - (actual\ start\ +\ laytime) \\ &= ETD - laycan\ end \\ cost &= h \cdot demurrage\ rate \end{aligned}$$

Where h represents the number of hours that the visit is handled after laycan end and *cost* represents the demurrage cost of the visit.

Case 4 in figure 7 shows a situation in which the visit is not handled or processed in a timely fashion. As can be seen in figure 7 the end of the post jobs task takes place after laycan end. Suppose the post jobs task has a duration of 4 hours, that means $h = 4$ and suppose the demurrage rate of this visit is \$200, the demurrage cost of this visit would be:

$$\begin{aligned} cost &= h \cdot demurrage\ rate \\ &= 4 \cdot 200 = \$800 \end{aligned}$$

Algorithm 2: Comparator function used to sort an array of visits

```
function compareDemurrage(a: Visit, b: Visit)

  if a.getDemurrageRate() !== b.getDemurrageRate() then
    return b.getDemurrageRate() - a.getDemurrageRate();
  end if
  if a.validLocations.length !== b.validLocations.length then
    return a.validLocations.length - b.validLocations.length;
  end if
  return b.duration - a.duration;
```

Algorithm 2 is used to sort an array of visits to form a preliminary order in which the visits should be considered to create a valid schedule or plan. The algorithm sorts the visits based on the demurrage rate of a visit in descending order because the visit with a higher demurrage rate should be planned first as the fine that would have to be paid if this visit were to be processed too slow would be higher per time unit than the other visit. The other heuristics that are used to sort the visits are the same as in algorithm 1. The goal is to minimise the demurrage cost, so in the example of figure 7, the visit has the best demurrage cost rating in case 1, 2 and 3.

So, for the berth time KPI, the sorting of the visits has no impact on the runtime of finding a valid schedule or plan. For the other two KPIs, the departure time KPI and the demurrage cost KPI respectively, a preliminary sorting has a significant impact on the rating of the plan that will be generated by our algorithm. The compare function for sorting the visits for the departure time KPI sorts the visits in ascending order based on the end or finish times first, if the end times of two visits are the same the algorithm considers the number of locations that the visit can be assigned to. Finally, if the number of locations is also the same the algorithm sorts the visits in descending order based on their respective durations. When optimising according to the demurrage cost KPI the visits are sorted based on their respective demurrage rate in descending order first, and afterwards it works the same as the algorithm for the departure time KPI.

7.2 Creating plans for multiple KPIs

Most of the time, the client wants a plan that is optimised for multiple KPIs. For example, a fast plan that costs millions is not a good plan for most clients. In this case, it is better to have a slightly slower plan that costs a lot less. This has been implemented in a similar manner as the algorithm for a single KPI, but with a rating function that combines all KPIs instead of using solely one KPI rating. The rating function has been defined as follows:

$$rating = rating_{demurrage} \,/\, ratio_{demurrage} + rating_{departure} \,/\, ratio_{departure} + rating_{berth} \,/\, ratio_{berth}$$

For $ratio_{KPI} > 0$, $ratio_{KPI}$ indicates the importance of KPI with regards to the other KPIs, where the value of $ratio_{KPI}$ is inversely proportional to its importance. However, a high value of $ratio_{KPI}$ does not always indicate that it is not important as this value is relative to the $ratio_{KPI}$ values of the other KPIs as well. This rating is used to compare all locations for the best placement of the visit. Every client may value KPIs differently, therefore these ratio-values can be tweaked to match the desired result.

7.3 Creating a valid schedule

Plans that are not valid are useless for the client. Dropboard uses custom rules to verify that a visit is valid. This makes visit planning more complex because there could be several hundreds of rules that every visit has to comply with. Luckily Dropboard already has a feature for planning a single visit. This function has a visit and a location as input and returns the earliest position that is valid on this location. This feature is used throughout the implementations in this project to ensure that a visit is planned on a valid location and time slot.

7.4 Post schedule optimisation

Some scheduling algorithms cannot guarantee an optimal solution to the problem. For these algorithms, some optimisations improve the schedule after the schedule has been made. Post-processing can solve sub-problems that the scheduling algorithm was not able to solve. This is possible by mutating the order of the visits and scheduling the visits in the resulting order.

For a generated plan, of which the order in which the visits were scheduled is known, all visits are evaluated to be delayed or not. A delayed visit is a visit that starts after its ETA. For each delayed visit, the cause is evaluated. A cause can be interpreted as the reason for a visit not starting earlier at the location it is scheduled on. There are three possible causes. Firstly, a visit may be delayed due to some rule (for example tidal fluctuations). Secondly, a visit may be delayed due to a restriction. Finally, a visit may be delayed due to another visit claiming the location.

If the cause is some rule, we do nothing. If the cause is a restriction that is unrelated to a visit or imposed by a visit that was not selected to be smart scheduled, we do nothing. If the cause is another visit and that visit was not selected to be smart scheduled, we do nothing.

However, if the cause was a restriction that was imposed by a visit that was selected to be smart scheduled, or the cause was another visit that was selected to be smart scheduled, we can make a swap. A swap means altering the order in which the visits are scheduled. The underlying idea is that if the delayed visit is scheduled before the cause visit is scheduled, it will no longer be delayed by the cause visit as it will claim the location before the cause visit is scheduled. For each instance of either of the two mentioned scenarios, the 'cause' visit's and 'delayed' visit's index are swapped.

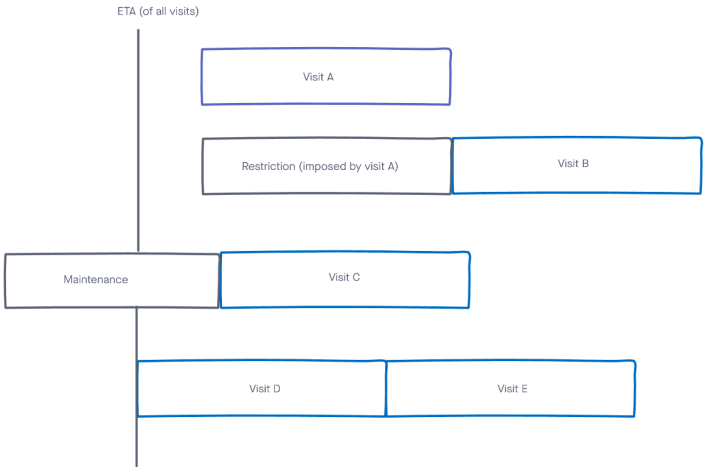

Figure 8: An example of visits and their delays

In the figure above, visit A is delayed due to some rule, visit B is delayed due to a restriction imposed by visit A, visit C is delayed due to a restriction that is not imposed by any visit, visit D is not delayed, and visit E is delayed due to visit D. Assuming all visits were selected to be smart scheduled, this would lead to a swap between visit A and B, and between visit D and E.

If at least one swap has been made, we run our algorithm with the new visit order. This yields a new plan. If the new plan was not better than the plan we already had, the algorithm stops and the 'old' plan is returned as the chosen plan. If however, the new plan is better than the old plan, this post schedule optimisation algorithm is called recursively, with the new (better) plan as input.

This will continue recursively until one of 3 things happens:

- There are no more delayed visits
- There are delayed visits, but none of them has a valid cause (thus no swaps are made)
- The newfound plan is not an improvement

In any of these scenarios, the 'old' plan is returned as the chosen 'best' plan.

7.5 Schedule single KPI

The first sub-problem, as indicated at the beginning of this section, is finding a schedule for a single KPI. Once an initial sorting of the visits has been made by the heuristic-based sorting part of the algorithm pipeline permutations of this ordering are considered by swapping two consecutive visits. (An overview of the entire pipeline is given in figure 5). For visits 1, 2, 3, and 4 this will look like:

$$[1,\ 2,\ 3,\ 4] \Rightarrow \{[1,\ 2,\ 3,\ 4],\ [\mathbf{2},\ \mathbf{1},\ 3,\ 4],\ [1,\ \mathbf{3},\ \mathbf{2},\ 4],\ [1,\ 2,\ \mathbf{4},\ \mathbf{3}]\}$$

The ordering on the left side of the arrow represents the heuristic-based sorted order of visits and the collection of orderings on the right side of the arrow represents the permutations of the heuristic-based sorted ordering that will be considered. By considering only swaps of adjacent visits, we end up with n permutations, where n is the number of visits. This is much better than for example the $n!$ permutations of the brute force approach. Swapping non adjacent visits would increase the number of permutations greatly, but is not likely to result in useful permutations, as the initial heuristic-based sorted order is most likely the optimal order. Consider the departure time KPI (see section 7.1.1). According to the heuristic-based sorted order, it can be concluded that visit 4 has a higher end time or visit 1 has a longer duration, thus swapping these visits probably will not impact the schedule positively. Therefore, permutations such as these are not worth considering.

For every permutation, each visit in that permutation is planned in that order. Each visit will be placed at its best location and time slot. This is ensured by calling the feature discussed in subsection 7.3 for every location that this visit can be placed on. This feature will provide the best time slot at a given location for a given visit. The visit is then evaluated on that location and time slot, for which a rating of the KPI is calculated, the location and time slot that provides the visit with the best rating for the KPI will decide its actual allocation.

This will result in multiple plans for the same KPI. For each of these plans, post schedule optimisation is applied, as explained in subsection 7.4. Consequentially, these plans are compared based on the overall rating of the entire plan for the corresponding KPI. The plan with the best KPI rating will be presented as the final plan for a given KPI.

7.6 Schedule multiple KPIs

The second sub-problem is finding a schedule for a combination of KPIs. The procedure for finding a schedule that performs well on multiple KPIs is the same as for finding a schedule for a single KPI, but the rating that is used is different from the ratings used for a single KPI. The rating used for this purpose has been explained in subsection 7.2.

8 Testing

8.1 Test Results

Throughout this project, automated tests have been written using the Jest framework. Unit, integration, and end-to-end tests have been written to verify the functionality of both standalone operations and a combination of operations. Also, tests determining the performance of the solution have been written. As a result of having written these tests, the coverage of our code looks like this:

File	% Stmts	% Branch	% Funcs	% Lines
src/multi-visit-smart-scheduling/brute-force	100	100	100	100
brute-force.ts	100	100	100	100
src/multi-visit-smart-scheduling/multi-visit-scheduling	94.12	85.11	93.75	93.1
multi-visit-scheduling-sorting.ts	93.18	84	100	92.5
multi-visit-scheduling.ts	100	100	100	100
schedule-finder.ts	90	80	66.67	88.1
weighted-KPI-scheduling.ts	100	66.67	100	100
src/multi-visit-smart-scheduling/post-schedule-optimisation	93.9	90.32	100	94.2
post-schedule-optimisation.ts	93.9	90.32	100	94.2
src/multi-visit-smart-scheduling/rating-calculators	100	100	100	100
all-kpis-calculator.ts	100	100	100	100
berth-time-calculator.ts	100	100	100	100
demurrage-cost-calculator.ts	100	100	100	100
departure-time-calculator.ts	100	100	100	100
src/multi-visit-smart-scheduling/single-visit-scheduling	98.04	76.92	100	100
allocate-visit.ts	88.89	50	100	100
get-valid-locations.ts	100	100	100	100
single-visit-scheduler.ts	100	71.43	100	100
src/multi-visit-smart-scheduling/test-utils	99.25	71.43	100	99.19
create-berth-valid-rules.ts	100	75	100	100
create-job-duration-rules.ts	100	83.33	100	100
create-visit.ts	100	65	100	100
test-utils.ts	97.87	80	100	97.83

The remainder of this section will discuss the performance of the solution and the comparison between this solution and the brute-force approach.

8.2 Performance

To determine how well the solution performs first a standard has to be set that serves as a baseline with which the solution can be compared. The standard in this situation is the brute-force approach in which all permutations of the visits are considered. This section discusses the performance of the solution in comparison to the brute-force approach.

8.2.1 Runtime

As mentioned before, the brute-force approach considers all permutations of the visits. So, for n visits the number of permutations that will be considered by the brute-force approach will be $n!$. By sorting the selected visits on the heuristics as mentioned in section 7, the number of permutations can be reduced.

As can be seen in figure 9, the number of permutations that will be considered drastically increases in the brute-force approach in comparison to this approach, called the smart scheduling approach. The number of permutations that will be considered in the smart scheduling approach linearly increases with the number

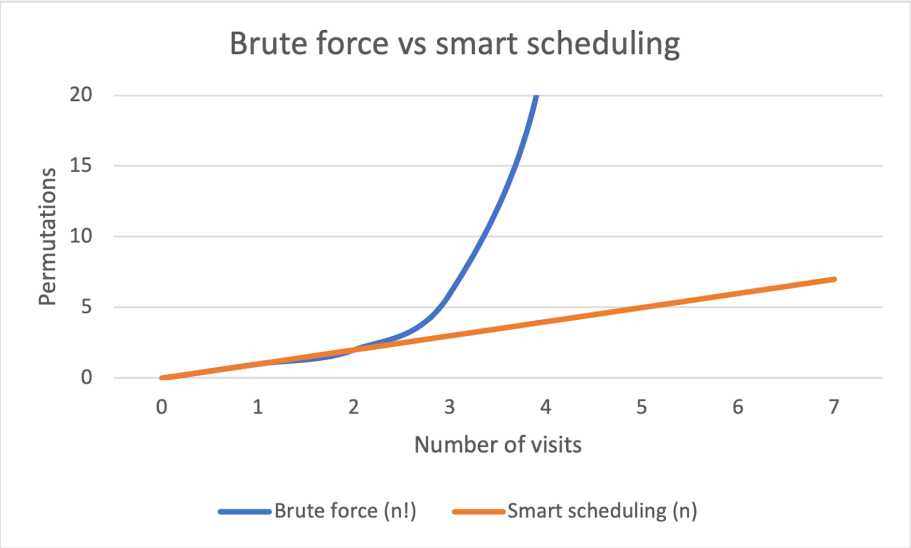

Figure 9: The number of permutations that will be considered

of visits. So, as the number of visits increases the difference between the number of permutations that the two approaches consider becomes larger.

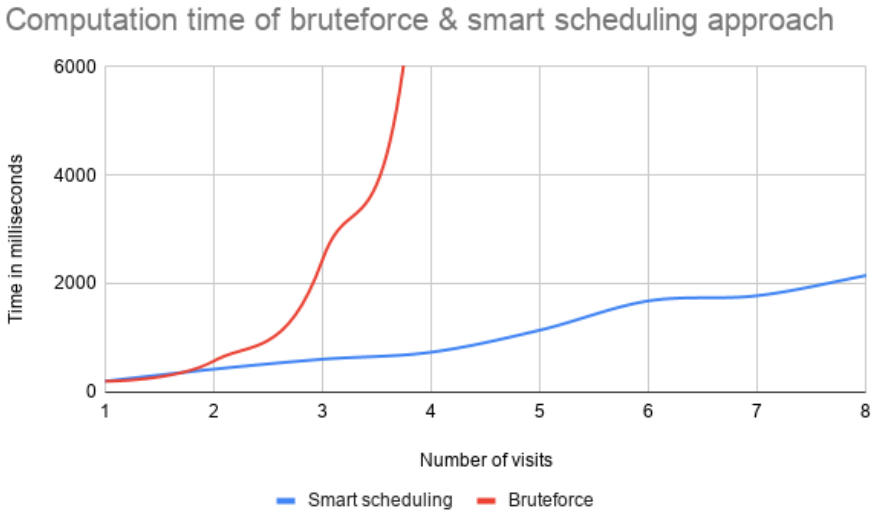

Figure 10: Time it took both approaches to generate a schedule for a single KPI

Figure 10 shows the runtimes of the situation in which there are visits that have to be planned on a single location and no rules are considered. The runtimes have been measured in a test environment, so it does not consider the amount of time that is needed to process unrelated elements such as the loading of UI elements, saving all visits in a schedule, etc. In reality, this of course also impacts the time it takes to finalise a schedule. Each approach has been ran 10 times with the same amount of visits such that an accurate average time needed could be determined. As can be seen in the figure above, even for this relatively simple situation, the brute force approach needs a large amount of time to generate a plan. For 4 or more visits, the brute force approach is no longer acceptable. The smart scheduling approach needs significantly less time to generate a plan. Up to 4 visits the schedule is even optimal in this scenario. However, this scenario is not an accurate reflection of reality as this does not take into account the vast amount of rules that are going to be considered when one of Dropboard's clients is going to use it. The reason this simplified situation is used to compare the two approaches, is that this way an accurate comparison of the runtimes could be made in an acceptable amount of time. What is also interesting, is the fact that figure 9 and figure 10 strongly resemble each other. From this it can be concluded that the number of permutations that are considered strongly influences the runtime of the algorithm.

8.2.2 Bottleneck

In our smart scheduling algorithm, we use a single visit allocation function that was already present in Dropboard. For each plan we find, it is invoked once for every visit on every location. This number possibly goes up due to post schedule optimisations. This function takes as input a visit, and allocates it on the best possible allocation on the location it is on. By doing so, it evaluates all rules in the environment to ensure the allocation is valid. Because of this, the execution of this function takes several hundreds of milliseconds. This duration, multiplied by the number of times we invoke this function, makes up the majority of the time our smart scheduling algorithm takes. Optimising the performance of this function was not in the scope of this project, so the only thing within the bound of this project was to invoke this function as few times as possible. This can be translated to trying as few permutations of visits as possible when generating a plan.

One idea we had was to use memoisation. So if the function is called for visit X on location Y, we would store this result in a cache and the next time this function was called for visit X on location Y, we would return the cached result. This turned out to be incorrect, as this function can give a different output based on other already scheduled visits, even if the input is the same. Consider the following situation: We are trying to schedule two visits, visit A and visit B. Furthermore, we have two locations, location 1 and location 2. Let's start by calling this function for visit A on location 1, and cache the result. Later on in the process, visit B is placed on location 2, and there is a rule in the system

that will delay visit A on location 1, because visit B is at location 2 during its ETA. If we would use the cached result, we would allocate visit A on location 1 in an invalid state, because of the mentioned rule. The only way to avoid this is to call the function again, which will consider all rules and other scheduled visits when allocating visit A on location 1.

9 Process

Communication is of vital importance in any project in which one has to collaborate with others. Particularly during these times in which we have to stay at home most of the time due to the COVID-19 pandemic. This section will provide an overview of our process during this project. First, an overview of our general week will be given, followed up by the methodology and tools we used.

9.1 Typical week

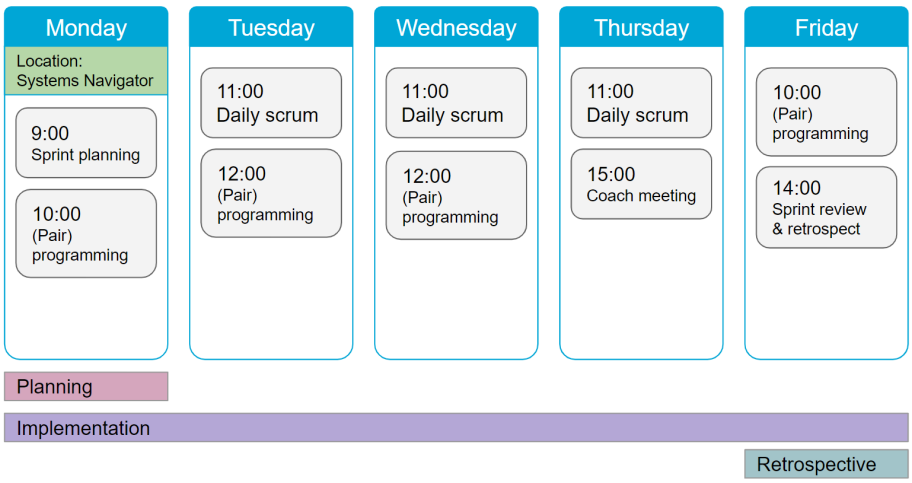

Figure 11: Overview of our typical week

During the first few weeks of the project, we could start off our week with a kick off meeting at the headquarters of Systems Navigator in Delft. This was especially helpful during the beginning of the project as the development environment had to be set up and we had to gain access to the codebase of Dropboard. It also helped in having clear project requirements as there was always someone from the client there to meet with us and explain things that were not clear. The setup of both the development environment and project requirements would have taken a lot more time if we had to do that online. This provided us with the unique chance to ask questions and immediately receive an answer, whereas online the communication would be delayed somewhat. Once the kick-off meeting was finished we went on to implementing solutions to the issues that had to be finished for the week. The other days of the week we would meet online as a group and discuss problems we were facing or the solutions we came up with the day(s) before. Every Thursday we would meet online with our coach from TU Delft to give an update regarding the project and ask questions

about problems we were facing. Finally, on Friday we would finish off our week by having an online retrospective meeting in which we reflected on the entire week and see what issues were solved and what issues still had to be done.

9.2 Tools and Methodology

Throughout this project the scrum methodology has been used, as can be seen in figure 11. At the beginning of every week we would have a kick-off meeting and every other day we would have an online daily scrum session in which we discussed what each of us has done. For these meetings, we used Microsoft Teams and for other communication within the group we used WhatsApp as we found that it was easier and faster to communicate that way. Microsoft Teams helped us with setting meetings with the client and TU coach as well as within the group itself. Furthermore, Mattermost was used to communicate with the project coordinators from TU Delft.

We have written most of our code in plane Typescript, the code for the UI elements of course also contains some HTML and CSS. We used the Angular framework for the UI elements. For the version control of our code, we used Gitlab as this was used by Systems Navigator. They provided us with an account and access to the part of their codebase that we had to work on. Using git helped with collaborating as everyone could work on their own parts, without having to wait for another person. We used Trello as our issueboard. Trello is a tool that helps to keep track of which issues have to be solved, who is working on which issues, which issues are ready for review, and which issues are done. The definition of 'done' in our case was when an issue had been reviewed and merged. We used draw.io for drawing diagrams to visualise the planning of visits and to support our explanations during the project. Finally, for writing this report we used Overleaf so we could work together in one document.

10 Discussions

Throughout this chapter, we will reflect on the process and mention points that can be improved upon and features that we would have liked to implement/ see in the product, but which were not requested by the client. First, we will reflect on the requirements, mentioning how well we have met them and possible changes. Followed up by the feedback we have received from the Software Improvement Group regarding the code we have written. Then, we discuss the ethical implications of this software and finally, we list our recommendations for how this product could be improved.

10.1 General discussion

We managed to implement the minimum requirements for the product (must haves) in the first two weeks of the development/ implementation phase. This included UI features as well as algorithmic features. During the project the client's requests slightly changed as they mentioned that some of their customers have something called 'user-defined business priorities' and a 'first come, first serve' guarantee. We had to adapt to this change in the requirements for the product. Luckily, we had designed the code in such a way that these changes could easily be added to the existing code.

Some of the should haves mentioned in the requirements (see Section 4.3) have changed slightly. The second should have has been replaced by the 'user-defined business priorities' and 'first come, first serve' features because opting out certain KPIs was less important, as the in house scheduler could always choose not to select the schedules that are generated for the KPI(s) the scheduler would otherwise want to opt-out. All of the could haves are implemented and none of the won't haves have been implemented.

As we could implement the changes in the requirements relatively easy, this indicates that our code is adaptable and can be used to solve various scheduling problems. However, there are some disadvantages to our implementation as well. These disadvantages are that the algorithm currently does not guarantee an optimal schedule, the algorithm relies heavily on the initial sorting of the selected visits and it only considers this initial sorting (with a couple of swaps), and the post scheduling optimisation algorithm we have implemented might stop at a local optimum.

10.2 Feedback Software Improvement Group (SIG)

There were two mandatory code submissions during this project. The first submission deadline was at the 75 percent mark of the development, and the second deadline was at the end. The written code was evaluated by the Software Improvement Group (SIG). The received feedback and our approach to improve our code will be discussed in this section.

10.3 First code submission

We received scores for multiple grading points. The scores were based on a scale up to 5.5, where a higher score means a better grade. The code that was submitted at the 75 percent mark was evaluated to be perfect at Duplication, Module coupling, and Component independence. Unit complexity was sufficient with a rating of 3.5, while the Unit size and Unit interfacing were poor with ratings of 2.3 and 1.7 respectively.

To improve Unit complexity, we had to reduce the McCabe complexity of two of our functions. One of these functions was a utility function that created a Rule, and the high McCabe complexity was because the function had 3 optional parameters that did not have defaults. We fixed this by adding defaults. The other function was our find schedule function, which had a high McCabe complexity due to nested for loops. We decided not to alter this function as the nesting was justified given the goal of the function.

To improve Unit size we split up functions into multiple functions. To improve Unit interfacing, we provided a single configuration object as an argument for a function rather than multiple separate parameters. Any new code we had written was also altered according to the mentioned countermeasures.

10.4 Second code submission

As a result of these changes, our grade for the unit size went up by 1.49 grade points from a 2.3 to a 3.7 and the grade for unit interfacing went up by 1.41 grade points from a 1.7 to a 3.1.

Our unit interfacing grade was still not perfect, due to some functions having 3 parameters, and very few functions even having 4 parameters. We used object destructuring to reduce parameters in most functions, but it seemed counterproductive in some other functions.

We have also tried to decrease unit size, but the largest negative contribution for this grade is the size of our scss file. The complete file is seen as one unit, which seems like a mistake. Either way, we did not know how to counter this, thus left it as is. Furthermore most units that were marked as too large were only a couple of lines too large.

In conclusion, by applying the changes mentioned above, our overall grade went up by 0.5 grade points from a 4.2 to a 4.7.

10.5 Discussion on ethical implementations

The software of Systems Navigator is going to be used by an in house scheduler of a terminal or port. The scheduler can adjust some parameters of our algorithm

to change the behaviour of our algorithm. These parameters can be tweaked to always choose the cheapest or the fastest option. This raises the question of whether this is always ethical. For example, a cheap ship could have to wait multiple days because there are a few more expensive ships that have priority. This could lead to some unethical situations that the scheduler has to deal with.

10.6 Recommendations

Currently, our algorithm only shows the best performing schedules according to some KPI rating. Suppose we are interested in a schedule for the departure time KPI our algorithm will select the schedule with the lowest overall departure time KPI rating (among the possible schedules it considers). This does meet the requirements of the client, however an improvement could be that the algorithm shows the scheduler multiple schedules for a single KPI. This way a schedule that might have a slightly higher overall departure time rating, but with a significantly lower demurrage cost rating might be of more interest to the scheduler than the one currently shown. Also, the product could 'learn' from the choices a scheduler makes regarding the type of schedules he or she selects, to determine a hierarchy in which the schedules should be shown to this client, and to possibly generate more of these type of schedules as they seem to be more interesting to this client.

As this project was very time restricted we could conduct limited research, so it is possible that other approaches might perform better than the one we have chosen. Therefore, we believe that conducting more research into scheduling techniques could be beneficial to determine whether exact approaches exist that can efficiently generate an optimal schedule.

Furthermore, our algorithm considers the number of valid locations for a visit, but we do not determine which locations are most popular, in order to avoid the allocation of visits to popular locations. This is a feature that could lead to better performing schedules, but at the cost of additional complexity. Based on this information a reinforcement learning model could be used to predict the best location for a visit during a specific time window.

11 Conclusion

During this project, we have explored the possibilities for an algorithm that can schedule multiple visits taking into consideration constraints and KPIs. The solution we came up with has been integrated into Dropboard and is ready to be used in production by Dropboard's clients.

Our algorithm takes as input the visits to be scheduled. It then sorts these visits based on heuristics and schedules each visit in this order on its best location, using the single visit/ single resource allocation algorithm that was already present in Dropboard. Multiple plans are generated and post schedule optimised to possibly improve their rating. These plans are shown to the user with their rating in each KPI, and the user can select them to view them on the chart. Finally, a user is able to select a plan and persist it into the operational plan.

Given the challenge this project posed and our rather small group size, we are happy with the final delivered product. Our fully functional algorithm has passed all requirements on the User Interface side, as well as performance and feature side. We believe that the algorithm will directly benefit Dropboard's clients as it adds a highly demanded feature. Through testing and demonstration, we are convinced that our product will perform at the high standard that Dropboard sets.

Besides being happy with the final deliverable product, we are also very happy with the process of this project. We were able to meet every deadline, both internally within the team, and externally. Our scrum approach proved to be successful and resulted in a clear but manageable schedule.

12 Reference list

References

[1] General guide for tu delft ti3806 computer science bachelor project (2020). https://brightspace.tudelft.nl/.

[2] Graph theory, part 2. http://web.math.princeton.edu/math_alive/5/Notes2.pdf.

[3] TU Delft. Dynamic programming: 6.1 weighted interval scheduling. https://ocw.tudelft.nl/wp-content/uploads/Algoritmiek_Weighted_Interval_Scheduling.pdf.

[4] Martin Charles Golumbic. volume 57 of *Annals of Discrete Mathematics*, page 171–202. Elsevier, 2 edition, 2004.

[5] Mart Jansen. Systems navigator - dropboard - advanced predictive planning. https://projectforum.tudelft.nl/course_editions/32/projects/923, 2020.

[6] Jon Kleinberg and Tardos Éva. *Interval Scheduling: The Greedy Algorithm Stays Ahead*, page 163–163. Pearson, 2014.

[7] Antoon WJ Kolen, Jan Karel Lenstra, Christos H Papadimitriou, and Frits CR Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[8] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2):174 – 190, 2011.

[9] Lalla Mouatadid. Algorithm design, analysis, and complexity: Dynamic programming, weighted interval scheduling. http://www.cs.toronto.edu/~lalla/373s16/notes/WIS.pdf, 2016.

[10] Dave Mount. Dynamic programming: Weighted interval scheduling. https://www.cs.umd.edu/class/fall2017/cmsc451-0101/Lects/lect10-dp-intv-sched.pdf, 2017.

[11] Frits de Nijs. *Project Scheduling: The Impact of Instance Structure on Heuristic Performance*. PhD thesis, 2013.

[12] Stephen Smith. Is scheduling a solved problem? *Multidisciplinary Scheduling: Theory and Applications*, 10 2004.

Appendices

A Course description

The TU Delft Computer Science Bachelor Project (also referred to as "BEP") is the final project carried out by Computer Science bachelor students. It is the last course of the undergraduate Computer Science curriculum. The bachelor project is carried out in a group of 5 students, who work together as a software development team. During the project, the team produces a product that is commissioned by a client and solves a real-world problem. The Bachelor Project also puts special emphasis on research skills: the students are expected to carry out research and demonstrate that they have arrived at optimal solutions, taking the full range of possible solutions into account. Over the course of the project, the student team experiences the entire trajectory of a real-world software development project, including:

- research and problem analysis
- requirements
- specifications and quality requirements
- implementation
- testing and validation
- delivery of a working product, including necessary documentation.

The team tackles this challenge by integrating their knowledge and experience from previous courses and projects in the curriculum. (Project description of the general guide [1])

B Info Sheet

Title of the project: Advanced Predictive Planning
Name of the client organisation: Systems Navigator
Presentation date: 25-01-2021

Project Description

Systems Navigator currently has a product called Dropboard. In Dropboard the clients can plan a single visit on the earliest available slot, this feature does not take into account constraints such as the effect one visit's allocation has on another visit. Thus, there is no support for scheduling multiple visits in a manner that accounts for the effects visit allocations can have on other visits and how those allocations influence the values or ratings of certain key performance indicators (KPIs). There are three main KPIs that are to be considered, the departure time, the berth time, and the demurrage cost. Furthermore, some clients offer their customers a first come first serve guarantee, which is also not supported. These limitations of Dropboard led to the creation of this project. Systems Navigator's request is to extend Dropboard's current user interface such that, based on a selection of multiple visits, multiple plans can be generated and shown to the user. These plans include plans that optimise according to a single KPI, a first come first serve schedule, and a plan that optimises according to the combination of the three KPIs. For each plan a short description of the respective plan should be given including the ratings of each KPI. After conducting research into scheduling techniques and how schedules can be optimised, we decided to opt for an algorithm using heuristic based sorting. Besides the heuristic based sorting, we use a post scheduling optimisation technique to possibly improve the schedules found with the heuristics based approach. Simulated annealing is used to decide how many iterations of the post scheduling optimisation algorithm are performed. Further research should be conducted into finding a more exact approach to generate an optimal schedule.

Team Members

Name: Sayed Mozafar Shah
Interests: Data Science & Engineering, Software Engineering, and Sports
Role and contributions: Front- & back-end developer and tester

Name: Ashwin Sitaram
Interests: Front-end web app development and Video Games
Role and contributions: Contact person, front- & back-end developer and tester

Name: Pepijn Klop
Interests: Software Engineering and Sports
Role and contributions: Front- & back-end developer and tester

Client and Coach

Name and Role: Mart Jansen, Product Owner (Systems Navigator)
Name and Role: Menno van Starrenburg, Mentor (Systems Navigator)
Name and Role: Matthijs Spaan, Coach (TU Delft)

Contact Person

Name and Email: Ashwin Sitaram, Ar.sitaram@hotmail.com

The final report for this project can be found at: `http://repository.tudelft.nl`

C Project plan

C.1 Introduction

For this graduation project we are tasked with finding and implementing a software solution that can be used by Systems Navigator's Dropboard clients so their in house scheduler can easily determine various possible schedules for a given day by clicking a few buttons. A good schedule aids in smoothing the workday. For any company it is vital to their success that every workday can run as smooth as possible with as few hiccups as possible. Planning ahead can help one anticipate issues or even avoid issues altogether, therefore this Project Plan, which is an agreement between us researchers, the client, and our coach, is set up. It will provide an overview of the project and its various phases.

C.2 Project Description

In this project, we will write a piece of software that can immediately be used in Dropboard. Firstly, the software will allow users to select multiple visits. Then, the software will generate multiple plans, that are different in performance regarding 3 key performance indicators. These KPI's are: departure time, berth time, and demurrage cost. Departure time is the time between a ship arriving at the port and leaving the port. This will include any idle time it spends during this duration. Berth time is the cumulative duration of each sub task of the visit. This will not include any idle times. Demurrage cost is the amount of money paid as a fine when the departure time of a ship is larger than a given range. This cost and range can differ per visit. When creating a plan, there are certain rules in place that can invalidate a visit. For example: certain ships can only go to certain Berths, due to size and cargo type. Another, more complex rule would be that if a ship larger than size X is at Berth A, then a ship of size $>$ Y cannot be at Berth B during the time the first ship is at Berth A. It is up to us to ensure all of our suggested plans are completely valid. Finally, once we have generated these plans, we will present them to the user. The user may view each plan, and select the one they seem to be the best. That plan should then be applied to the operational plan in Dropboard.

C.3 Deliverables

Throughout this graduation project the following mandatory deliverables are supposed to be handed in.

- Project Plan (20-11-2020)
- The current document, a general overview of the graduation project.

- Research Report (04-12-2020)
- A report listing the findings of the research phase (first two weeks). This report will also determine the techniques to be used in the software solution.
- Final Report (18-01-2021)
- "Description of the final report here"
- Info Sheet (18-01-2021)
- A single A4 page summarising and highlighting the entire project.
- Code review submissions (evaluated by the Software Improvement Group (SIG) at TU Delft)
- First code submission: (20-12-2020)
- Second code submission: (18-01-2021)
- These submissions should contain all the code that we are allowed to share and have written up to the indicated dates.
- Final Presentation (25-01-2021)
- A 30-minute presentation including a demo of the product. Followed up by questions of the course coordinators, our coach, the client, and possibly other people watching.

C.4 Project Roadmap

C.4.1 Week 1-2 Research Phase

During the first two weeks the sole focus of the project will be to perform research into the domain of the given problem. The goal of this research is to determine the different scheduling techniques or algorithms and their use cases. Based on these findings a comprehensive analysis will be made to determine which of those techniques are most suitable for Systems Navigator. These findings and their corresponding analysis will be documented in the Research Report.

C.4.2 Week 3 Investigation Phase

Following the research phase, it is necessary to evaluate the selected scheduling algorithms and find out whether any modifications to the algorithms are needed, to find the limitations of those algorithms with regards to their use cases, and the compatibility with Dropboard's current software. These factors weigh in to the process of estimating the time needed to implement and verify the algorithms. In conclusion to

this phase a somewhat permanent selection of scheduling algorithms has to be made and a plan for implementing and modifying the respective algorithms should be in place.

C.4.3 Week 4-6 Implementation Phase

Having selected the algorithms and decided how they should be implemented, during this phase the actual implementation of those algorithms will take place. If any modification to the algorithms have to be made to better fit the purposes of Dropboard, that must also happen during this phase. Additionally, implementation of tests to validate these algorithms happens in this phase too.

C.4.4 Week 7-8 Modification and Expansion Phase

At the end of the implementation phase at least the minimum requirements of the clients are met. Throughout this phase the goal is to either modify the existing code to optimise the algorithms or to add other features requested by the client.

C.4.5 Week 9-10 Report and Presentation

During this phase last tweaks to the software will be made (if needed). The Final Report and Presentation will also be finished during this phase. At the end of week 9 a draft version of the Final Report is supposed to be finished, so any feedback received by the coach can be added to the final version of the Final Report that has to be handed in in week 10.

C.5 Communication Plan

As a result of the ongoing COVID-19 pandemic we will mostly work from home. Communication within the group will mostly take place on WhatsApp and Microsoft Teams will be used for meetings and sharing of important files. Microsoft Teams is also used for communication with the client. Weekly meetings will take places in which we discuss our progress and any problems we have encountered. These meetings also provide us with the opportunity to check whether we are all still on the same page regarding the request of the client. We have a weekly meeting with our coach, usually planned on Thursdays, to give him an update on our progress and so we can ask him any questions if necessary. We use Microsoft Teams for these meetings as well. The project will be executed using the agile approach. Every week we will have (at least) two meetings within the group, the first one on Mondays to set new goals and discuss upcoming deadlines. The second one, which serves as a retrospective meeting, will take place on Fridays.

C.6 Tools

- Communication Team:
  - Whatsapp
  - Microsoft Teams
  - Mattermost
- Communication with Systems Navigator:
  - Outlook
  - Teams
- Communication with coach:
  - Teams
- Code:
  - Gitlab
- Issue board:
  - Trello
- Programming languages:
  - Typescript
  - HTML/CSS
- Framework:
  - Angular
- Paper:
  - Latex Overleaf

D Glossary

Arrival: The visit's task representing the ship sailing into the port or terminal, to a location.
Berth: A large location that a ship can be assigned to, to unload its cargo or load new cargo.
Berth time: The duration of all tasks of a visit accumulated.
Demurrage: A fine that has to be paid by the port if a visit is not processed within a defined window of time.
Departure: The visit's task representing the ship sailing out of the port or terminal.
Departure time: Number of hours a ship departs after its ETA.
ETA: Estimated Time of Arrival of a ship. Depicts the date and time at which a ship will be arriving at the port or terminal.
Invalid plan: A plan in which some visit violates a rule.
Jetty: A smaller location than a berth, serves the same purpose as a berth but typically for smaller ships.
Job: The visit's task representing the (un)loading of cargo. A visit may have multiple jobs.
KPI: Key Performance Indicator.
Laycan start/end: The start and end of the window in which a visit has to be processed such that no demurrage cost has to be paid by the port.
Laytime: The duration of the laycan window, so the difference between laycan end and laycan start.
Liquid bulk terminal: A part of a port focused on processing crude oil and oil products, liquid chemicals and edible products.
Moored: A ship securing itself to a location, so that it can be processed and will not float away.
Plan/Schedule: An arrangement of visits, where each visit is assigned a location and all of its tasks are assigned a start and end date (and thus a duration).
Port: A group of locations aimed at processing cargo of ships.
Rule: A rule consists of a number of conditions and a specification of what should happen. With rules, almost any type of restriction can be enforced on visits.
Scheduling technique: Algorithmic technique used to determine a schedule.
Visit: A representation of a ship 'visiting' the client's port or terminal in Dropboard.

E Proprietary figures

The figures in this section are screenshots of Dropboard. These screenshots are proprietary information and may not be shared in any way. These are for the coach and client only, and will not be included in the version of this report that will be shared on the public repository.

Figure 12: Selecting visits to be smart scheduled

Figure 13: Smart scheduling in process

Figure 14: Showing the best found plans