Optimization of crude oil operations scheduling and product blending and distribution scheduling within oil refineries

van der Hoek, T.

Award date:
2014

OPTIMIZATION OF CRUDE OIL OPERATIONS SCHEDULING AND PRODUCT BLENDING AND DISTRIBUTION SCHEDULING WITHIN OIL REFINERIES

TOM VAN DER HOEK

Master's Thesis

Department of Mathematics and Computer Science
Architecture of Information Systems
Eindhoven University of Technology

September 2014

Tom van der Hoek: *Optimization of Crude Oil Operations Scheduling and Product Blending and Distribution Scheduling within Oil Refineries,* Master's Thesis, © September 2014

SUPERVISORS:
Prof. dr. ir. W. Nuijten
ir. C. Verberk
MSc. R. van der Velden

LOCATION:
Eindhoven

TIME FRAME:
September 2014

ABSTRACT

Oil refineries are faced with complex scheduling problems which have seen several decades of research. To reduce the complexity of the to be solved scheduling problem, the oil refinery process is generally split into three separate processes. These three processes are *Crude Oil Operations*, *Component Production Process*, and *Product Blending and Distribution*. This thesis studies the Crude Oil Operations Scheduling Problem (COSP) and the Product Blending and Distribution Scheduling Problem (PBDSP) on behalf of the Business Unit Oil and Gas (BUOG) inside software vendor Quintiq [2].

The COSP starts at the unloading of crude oil, after which the oil is stored in storage tanks, blended in charging tanks, and finally fed to a crude oil distillation unit. This problem has several non-linear constraints which increase its complexity. To solve the COSP we study the best known solution method in literature, which is the solution method proposed in Mouret 2010 [15]. Here a two-step approach was proposed, that uses a Mixed Integer Programming (MIP) model for the first step and a Non-Linear Programming (NLP) model for the second step. In this thesis we present an improved MIP model for the first step, which is up to 9 times faster than Mouret's model and as such allows to solve instances that up to now were beyond reach. To solve the NLP problem of the second step, we study several approaches and show that the NLP solver CONOPT [1] performs best.

We also propose a heuristic model that finds a solution in one step. This *1-Step Heuristic* replaces the non-linear constraints by linear constraints that only allow for cases in which the non-linear constraints are satisfied. This model has the advantage of being linear, so a non-linear solver is not needed. The experiments show that it finds good results within a practically acceptable time limit.

The optimizers were tested using real life examples taken from literature and larger examples inspired by Quintiq's practice [2]. The final recommendation to Quintiq is to use the two-step approach using our improved MIP model together with the NLP solver CONOPT.

The PBDSP is about the blending of *components*, which are intermediate products made in the Component Production Process, into final products, and the distribution of these final products. We extend the PBDSP, compared to what is known in literature, by allowing for the scheduling of loading and unloading through the use of pipelines. This extension was a requirement coming from Quintiq's practice.

The PBDSP is a combination of two separate problems, namely i) *recipe optimization*, which finds the best blend of components for a

product, and ii) *logistics optimization*, which solves the logistic problem of moving the components and products through the network.

We present two approaches to solve the PBDSP. The first is a combination of iterative *Linear Programming* and iterative *Mixed Integer Programming*. The second approach uses blending indices to keep the model linear without the need to use an iterative approach.

The optimizers are again tested on instances inspired by Quintiq's practice. These are new instances as the PBDSP, as defined in this thesis, has thus far not been solved in literature before. The results show that the problem is solved close to optimality within a practically acceptable time limit, by both approaches, but we see that the second approach has the better performance.

Mostly, when you see programmers, they are not doing anything. One of the attractive things about programmers is that you cannot tell whether or not they are working simply by looking at them. Very often they are sitting there seemingly drinking coffee and gossiping, or just staring into space. What the programmer is trying to do is get a handle on all the individual and unrelated ideas that are scampering around in his head.

— Charles M. Strauss

ACKNOWLEDGMENTS

This master thesis concludes my graduation project performed during my study of Business Information Systems at Eindhoven University of Technology (TU/e). This project was made in collaboration between TU/e and Quintiq. I highly valued the contribution of several people during this project and I would like to thank them.

First of all I like to thank my graduation supervisor Wim Nuijten. He supported me greatly during the project and pushed me to aim as high as possible. I also greatly valued the feedback he continuously gave me on my thesis. Furthermore he came to me with the possible career opportunity and graduation assignment at Quintiq for which I am extremely grateful. Next I like to thank my Quintiq supervisor Coen Verberk and the other Business Unit Directors of the Business Unit Oil and Gas for allowing me to do one of their research projects. I also want to thank my Quintiq mentor Ronald van der Velden who guided me on using the Quintiq software. I also like to thank the rest of my new colleagues at Quintiq who welcomed and supported me.

Finally, I like to thank my parents, family and friends for their support during the project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

We start the introduction with a short motivation for the thesis in Section 1.1. Section 1.2 introduces the company where the study took place. Next the research assignment, methodology, and contributions of this thesis are given in Sections 1.3-1.5. Section 1.6 takes a look at related work and Sections 1.7 and 1.8 introduce several subjects that facilitate the understanding of this thesis. Finally, Section 1.9 gives an overall overview of the structure of the thesis.

1.1 MOTIVATION

Oil refineries have a large supply chain process. When this process is optimized it can save the refinery millions of dollars [10]. Because of this there is and has been a lot of research on optimizing the oil refinery process.

Despite the research done to date, there is limited useful work for this thesis, which is because of two reasons. The first reason is that quite some work created inefficient optimizers, meaning that they give optimal solutions for real life instances but are too slow for the intended purpose of this thesis, see for instance the work of Lee et al. 1996 [11]. The second reason is that research makes unrealistic assumptions, such as fixed blending recipes or linear blending functions, see for instance the work of Jia and Ierapetritou 2003 [8]. There are however some research papers which are highly relevant and are used as inspiration and starting points to solving the problems of this thesis. These papers are primarily Mouret 2010 [15] and Méndez et al. 2006b [13].

The reason a lot of optimizers are inefficient or make unrealistic assumptions is because the problem has complexities that cannot be solved easily using today's technology. Because the entire problem is so complex it is often split into three subproblems, namely: *Crude Oil Operations*, *Component Production Process*, and *Product Blending and Distribution*. This thesis will focus on solving the *Crude Oil Operations Scheduling Problem* (COSP) and the *Product Blending and Distribution Scheduling Problem* (PBDSP). We look at the scheduling problems because it includes all the complexities of the problem. Scheduling means that we want to create a detailed schedule where we take precise decisions on when to start and end the operations in the refinery and on the volume transferred during these operations. This implies that the scheduled tasks have a start, end and volume given to them, in such a way that the schedule can be executed by the refinery op-

erators. This is mostly done for small horizons as a lot can change in the real world which can make that a schedule becomes practically unusable.

1.2 QUINTIQ

The study was done on behalf of the *Business Unit Oil and Gas* (BUOG) inside Quintiq [2].

Quintiq is a company that sells software that solves planning and scheduling problems. The software uses optimization technology to create plans and schedules automatically. Quintiq is active in a lot of verticals (industry and/or region based), like the metals industry or aviation. One of the newer verticals is the oil and gas industry. As Quintiq is relatively new to this vertical, there are optimization problems that have not been solved yet within Quintiq. Some of these problems lie within oil refineries. BUOG is continuously improving their planning and scheduling software for oil refineries, by doing in-house research. This thesis is part of that research, where we look into the COSP and PBDSP to see if they can be scheduled automatically using Quintiq's software and optimizer technology. This brings us to the research assignment defined in the next section.

1.3 RESEARCH ASSIGNMENT

The research assignment of the project is formulated as follows:

- Create optimizers for the COSP and for the PBDSP that give a good solution in reasonable time. The BUOG defined a solution to be good if the obtained objective value is at most 5% from the optimal objective value, and the CPU time to be reasonable if it is at most 5 minutes. Priority is given to *Mathematical Programming* technology and *Constraint Programming* technology as these technologies are readily available in Quintiq.
- Integrate optimizers in a Quintiq application, so that the obtained solutions can be validated to be correct.
- Apply the optimizers on instances known from literature and realistic instances inspired by Quintiq's practice to show the quality of the optimizers.

Now, as progress came faster than anticipated, the following goal got added over the course of the project:

- Develop a better solution method than the best solution method known in literature for the COSP.

1.4 METHODOLOGY

To complete the research assignment the following methodology is used:

1. Problem definition. We first start with the problem definition phase. Crucial components are determining input data, defining what a solution to the problem is, determining the constraints defined on solutions, and defining the *Key Performance Indicators* (KPIs) of the problem. One tool we use is to build a set of small test instances to increase our understanding of the constraints and KPIs.
2. Developing optimizers. In this phase mathematical models are created, which serve as the basis to develop optimizers, to solve the defined problem.
3. Benchmarks. In the benchmark phase test instances are found and/or created to test the implemented optimizers.
4. Iterative phase: The different test instances (may) pinpoint shortcomings of the optimizers, which we improve in an iterative way. As we go we will do extensive testing to determine which properties of the optimizers contribute to their strength in what way.

1.5 CONTRIBUTIONS

The main contributions of this thesis are:

- We took the model presented in Mouret 2010 [15] and found several changes and additions which improved the performance. We also take a look at cardinality constraints, show how important they are, and devise a rule to get the input data needed for these constraints.
- An efficient way of solving the PBDSP, which includes the loading of final products, and unloading of bought components using main pipelines. We use an (iterative) LP/MIP model, which uses the model found in Méndez et al. 2006b [13] as a basis and is extended to allow for precise distribution scheduling. This allows Quintiq to model and solve the PBDSP in practice.
- For the COSP the standard set of benchmarks known in literature of 4 instances is extended by i) 3 new and larger instances, which also have different types of vessels, and ii) 13 variants on the now 7 main instances.
- We took the known MIP + NLP approach from literature, see Mouret 2010 [15], and developed an equivalent MIP + CP approach having similar solutions on the original 4 benchmark instances.

- A MIP that solves the COSP using a non-linear avoidance approach, meaning that the non-linear constraints are replaced by linear constraints that only allow for cases in which the non-linear constraints are satisfied. It has the advantage of being a linear model, so a non-linear solver is not needed.

1.6 RELATED WORK

The planning and scheduling of oil refinery operations has seen several research papers over the last decades, though it seems the research is on the decline as less work is found when looking into the last ten years. An overview of past research can be found in Bengtsson and Nonås 2008 [5].

The research on the COSP is mainly focused on short-term scheduling. There are two main approaches to solve this problem. The first is creating a (successive) *Linear Program* (LP) [4], which means that the non-linear constraints are relaxed. To still get close to a usable solution, the optimizer is made successive, getting closer with every iteration. To do this, special relaxation techniques are used like *McCormick Envelopes* [12], a technique that changes a non-linear constraint into several linear constraints that give a bound that can be tightened in every iteration. The second approach is decomposition of the *Mixed Integer Non-Linear Program* (MINLP) into a *Mixed Integer Program* (MIP) and a *Non-Linear Program* (NLP) [15]. This approach is used as the MINLP cannot be solved in a reasonable time with present technology. Using this decomposition, first a MIP will be solved which can be done very fast. Then using the solution of the MIP, an NLP can be defined where several variables are already bounded due to the MIP solution. This NLP can then be solved, using the non-linear solver CONOPT, within a second according to the results presented.

There were only several papers found on the PBDSP problem and nothing very recent, this may be because research is kept confidential or the research focus is more on the other two processes.

Glissmann and Gruhn 2001 [7] presents a solution method that uses an NLP to do recipe optimization and use the solution as input to a MIP that solves the scheduling problem. Jia and Ierapetritou 2003 [8] gives a MIP model to solve the scheduling problem but assumes fixed recipes, which is not always the case or wanted. Méndez et al. 2006b [13] presents a successive LP/MIP that solves both the recipe optimization and the scheduling problem in an efficient way, but does not precisely schedule the final distribution of the demanded products to the vessels or the unloading of bought components.

1.7 OIL REFINERY

An oil refinery has as main goal to produce final oil products such as gasoline, diesel fuel, asphalt base, heating oil, and kerosene. It is normally a large facility with large storage tanks for crude oil and products ("products" is often used as shorthand for final oil products) and distillation units to distil crude oil. Extensive piping runs between the tanks and to and from the distillation units to transfer the crude oil and products. The process that goes on inside such a refinery will be explained in this section. An overview of the entire process is shown in Figure 1.

As in practice it is found that the entire process is complicated to optimize, it is generally split into three sub-processes. The first process is called *Crude Oil Operations* which covers the unloading of crude oil, the transfer of oil between the tanks, and the transfer of crude oil to *Crude Oil Distillation Units* (CDUs). The second process is called *Component Production Process* which starts at the CDUs and ends at the component storage tanks. This thesis will not go into depth on this second sub-process as it is outside the scope of this study. This was decided as it is a more complex and different problem compared to the other two. The third and last process is called *Product Blending and Distribution*, sometimes also referred to as *Product Blending and Shipping*.

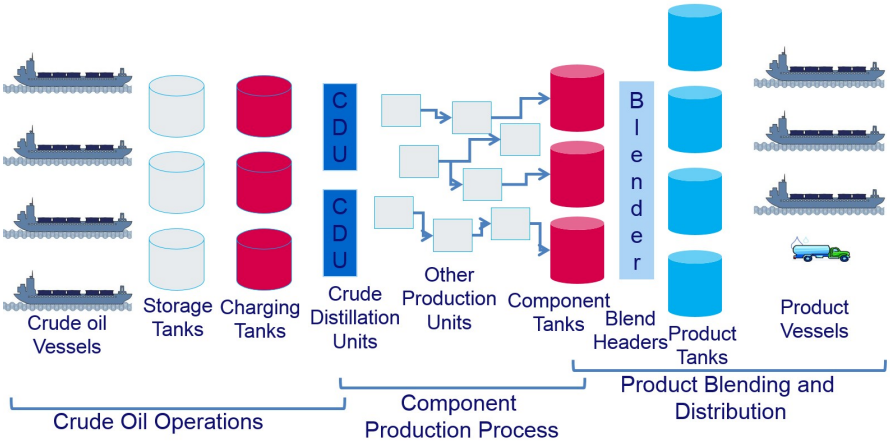

Figure 1: An overview of an oil refinery process

1.7.1 *Crude Oil Operations*

The Crude Oil Operations Process starts with the supply of crude oil. The crude oil comes to the oil refinery through oil tankers or via a direct pipeline. There are generally two types of oil tankers. The very large crude carriers (VLCCs) that can carry multiple crude types and smaller vessels that carry only one type of crude ("crude" is often

used as shorthand for crude oil). The VLCCs are normally too large to come close to shore so they dock at a single buoy mooring (SBM) station offshore. The smaller vessels berth at a jetty (smaller docking place). From both types of docking stations the oil is unloaded via a pipeline and stored into large storage tanks. In most cases there will be only one SBM, thus the VLCCs could have a queue as only one can unload at a time.

After the crude oil has spent some time inside the storage tanks to let the brine (sea water) settle, which may take several hours, it may be further processed. The crude oil will next be transferred from the storage tank to a charging tank. A charging tank may get several types of crude oil to blend. The blending may be done to get crude oil that can be better processed by the Component Production Process.

The blending of crude oil is one of the operations that make oil refineries complex. Crude oil has several properties and characteristics, for instance viscosity or the sulfur percentage. The values of these properties and characteristics are used to define the quality of the crude oil / crude oil mix. The values of the properties and characteristics of a mixed crude oil obviously depend on the values of the properties and characteristics of the used crude oils. The difficulty is that the dependencies between the values is not always linear. For example 20 bbl (oil barrel) of crude oil type A mixed with 40 bbl of crude oil type B does not have the same quality as 10 bbl of crude oil type A and 20 bbl of crude oil type B.

When a charging tank is ready, it may start discharging the crude oil to a CDU. As the Component Production Process is a continuous process, the CDU needs to be fed continuously from a charging tank. In most cases only one charging tank can charge a CDU at a time, and a charging tank can only charge one CDU at a time.

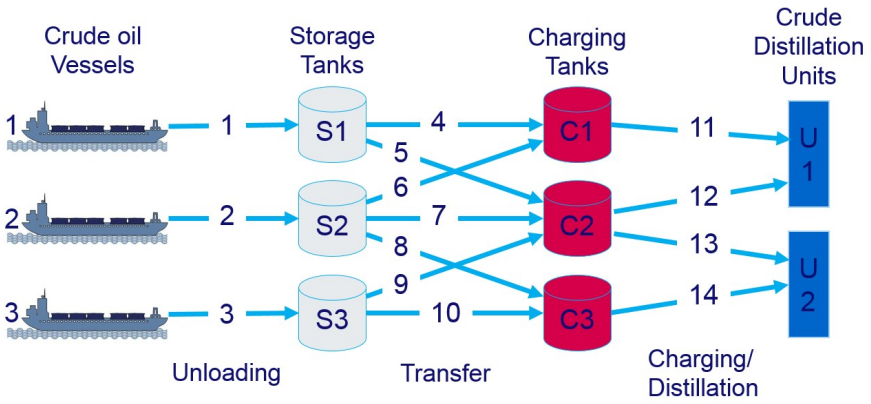

Figure 2: Overview of the crude oil operations process from Mouret 2010 [15]

1.7.2 *Component Production Process*

The central equipment in the Component Production Process are the CDUs. An illustration of a CDU is given in Figure 3. A CDU gets crude oil fed from a charging tank. The crude oil inside the CDU is then heated and becomes gas and floats upward, or is too heavy and goes down, through the distillation tower. In gas form all the components of the crude oil can be separated by their condensation point. The components may then be further processed by production units that complete the needed components. These units include cracking unit, alkylation unit, and reformer. All the components are finally stored in their own storage tanks.

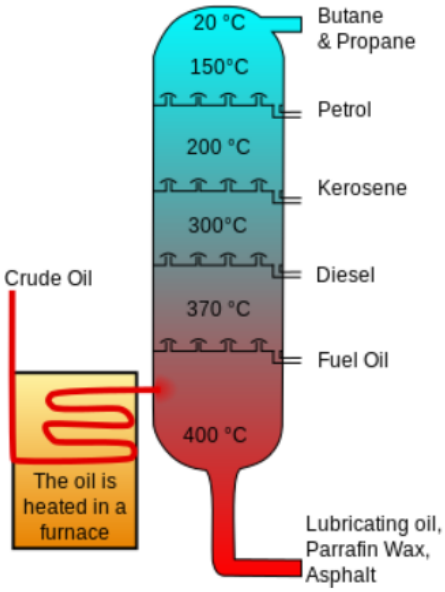

Figure 3: An illustration of a CDU from Wikipedia (2014)

1.7.3 *Product Blending and Distribution*

Some of the components coming out of the Component Production Process are final products and can directly go to finished product tanks. Others need to be blended with other components in a *blend header*. A blend header mixes the components in real time while they are fed through the pipeline without intermediate storage. It keeps track of the blend properties and can make small adjustments to keep the outflow inside the product specifications. Thus the big difference between a blend header and a blending tank (charging tank) is that a blend header constantly needs input that keeps the blend within specifications as there is no storage. Some of the properties have non-linear blending behaviour which makes this a complex problem to solve.

The blending of products can be seen as a separate problem, called *recipe optimization*. This problem ignores the logistic constraints and only looks at finding the best blend of components per product. The best blend is the blend with the highest value within the given product specifications. The solution is a *product recipe* per product. A product recipe states the fraction per component for the product. An example of recipe optimization is given in Figure 4. In the figure we see three different components $C1$, $C2$, and $C3$, with their respective cost and property value for property $K1$, a blend header $B1$ which can blend the components, and product $P1$ with its property specification. As recipe optimization only looks at product specifications we do not need any information about tank capacities, initial tank levels, etc. Recipe optimization simply looks at the cheapest way to blend 1 bbl of product $P1$ using the three components, while adhering to the product specifications. As we can see in the figure, the optimal recipe for product $P1$ is then to let $3/4$ of the total blended volume consist of component $C1$ and $1/4$ consist of component $C3$.

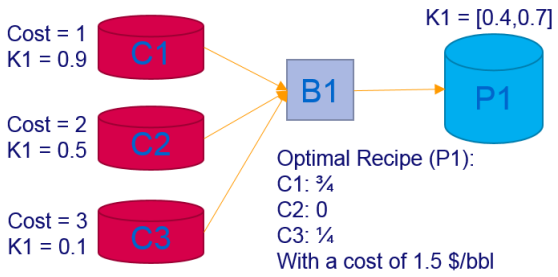

Figure 4: An example of recipe optimization

These optimal product recipes can then be set as a *fixed recipe*. A fixed recipe means to always blend the product according to the recipe, which means that the product specifications are not needed anymore at this point. Thus truly separating the blending and logistics problem. Another possibility is using the recipes as preferred recipes, thus still allowing for deviation from the recipe. If the quality is optimized during the scheduling, with or without using preferred recipes, the problem becomes more complex, but could also give better solutions.

After the blend header blends the different components into an end product it is transferred to product storage tanks. From these product storage tanks the products will be distributed to their respective vessels.

The distribution from the product storage tanks to the vessels uses a set of main pipelines. These main pipelines are used for loading the product inside one product storage tank into one vessel that demands that product. There is also the possibility of buying components from outside sources. These will then be delivered by *component vessels* and

will use the same pipelines as for the loading to unload the component from the vessel into a designated component tank.

1.8 OPTIMIZATION TECHNOLOGY

Optimization Technology is defined as technology used to solve *combinatorial optimization* problems. In a combinatorial optimization problem one is given a set of *decision variables*, a set of *constraints*, and an *objective function* (also called *goal function*). The problem is to find an assignment of values to decision variables, called a *solution*, such that:

1. all constraints are adhered to (i. e., the solution is *feasible*)
2. the objective function is optimized (i. e., the solution is *optimal*)

A solution is optimal when there can be no other solution that has a higher (lower) value for the objective function, when the problem is being maximized (minimized). An example of a combinatorial optimization problem is the following:

x, y, z (decision variables)
maximize $x - y * z$ (objective function)
subject to:
$1 \leq x \leq 10$ (1) (constraints)
$1 \leq y \leq 10$ (2)
$1 \leq z \leq 10$ (3)
$x - y > 6$ (4)
$z + y < 5$ (5)
$x + y > 7$ (6)
$z * x > 18$ (7)
Solution: $x = 10$, $y = 1$, $z = 2$, optimal value is 8.

Optimization technology has been used for decades to solve supply chain problems. For many problems optimization technology is capable to not only find a feasible solution but also finding a feasible optimal solution. There are several different optimization technologies such as for example *Mathematical Programming* (MP), *Constraint Programming* (CP), and *Local Search* (LS).

Combinatorial optimization problems can be classified into different groups. In this thesis we use four different groups namely: *Linear Programming* (LP), *Non-Linear Programming* (NLP), *Mixed Integer Programming* (MIP), and *Mixed Integer Non-Linear Programming* (MINLP). An LP problem is a problem that is defined by variables with continuous domains (i. e., reals) and linear constraints, meaning that the

constraints can be written as linear functions of the variables. Examples are constraints 4, 5, and 6 in the example above. With an NLP problem there are also non-linear constraints, for instance constraint 7 in the example. A MIP will have variables that have binary or integer domains, but can still have variables with continuous domains, and will only have linear constraints. Lastly there is MINLP which is a combination of NLP and MIP.

It is important to note that LPs are in the computational complexity class P where MIPs are in NP thus making LPs structurally easier to solve. For a detailed discussion of complexity classes we refer to Garey and Johnson 1979 [6]. Note that when there are non-linear constraints involved, the practical complexity to solve a problem is increased which is why such constraints are avoided whenever possible.

When using MP the example above can be used as input and the solver will search for the optimal solution using mathematical methods. CP works a bit differently. First of all CP only has decision variables with an integer or binary (boolean) domain, the reason is that CP needs finite domains to work with. When a CP starts searching for a solution it does essentially two things. The first thing is constraint propagation, which means that the domains of the variables are set tighter using the constraints. Of course there are a lot of problems where this is not enough to decrease all the domains to one value. This means that a search is needed which is the second thing. The search is simply choosing a value for a decision variable within its domain after which constraint propagation can happen again to tighten the domains further. The search follows a *search strategy*, this strategy says which variables should be chosen and which value in the domain should be looked at. An example of a search strategy is for instance: pick the largest possible value of the variable with the largest domain. If a smart strategy is used then the search can be reduced significantly.

1.9 STRUCTURE

The remainder of this thesis is split into two parts. The first part I has as subject the COSP, and the second part II has as subject the PBDSP.

Both parts are structured in the same way. They start with a chapter on the problem statement of their respective problem, see Chapters 2 and 6. Followed by a chapter about the solution methods, see Chapters 3 and 7. Then the different solution models are presented in Chapters 4 and 8. Finally, there is a chapter on the computational study of the respective problem, see Chapters 5 and 9.

The thesis is then finalized with Chapter 10, which presents the overall conclusion and possible future work.

Part I

Crude Oil Operations

2 PROBLEM STATEMENT CRUDE OIL OPERATIONS

This chapter describes the Crude oil Operations Scheduling Problem (COSP) and provides an abstract model of this problem. The problem description and abstract model are inspired by Lee et al. 1996 [11] and Mouret 2010 [15].

2.1 GENERAL DESCRIPTION

The central activities in Crude Oil Operations Scheduling are the *unloading*, *transfer*, and *charging* (also called *distillation*) *operations*. An unloading operation unloads crude oil from a crude vessel to a storage tank. A transfer operation transfers crude oil from a storage tank to a charging tank. A charging operation charges (transfers) crude oil from a charging tank to a CDU. The COSP can be described as determining when operations execute (this can be several times during the schedule), how long each execution takes, and how much volume is transferred during an operation, such that the total gross margin is maximized. The total gross margin is defined by the sum of crude volumes that are transferred into a CDU times their respective gross margin.

A COSP instance is defined by the following input data:

- A scheduling horizon, which indicates the ending of the schedule.
- Arrival times of the crude vessels, indicating when the vessels arrive.
- A vessel type for each vessel, indicating the berth the vessel will use. We assume that we have exactly one berth for every given type. If there is no vessel type given in the input data, then we assume they are all the same type.
- Capacity limits of tanks, indicating how much a tank can hold.
- Initial composition of vessels and tanks, indicating i) which crude oil type is present at the start of the schedule and ii) how much crude oil is present.
- Gross margins and property values for every crude type, indicating the expected profit of distilling the respective crude type and what the value is for a certain property for the respective crude type.

- Demands for each charging tank, indicating how much volume needs to be transferred using charging operations during the schedule.
- Crude property specifications per charging tank, indicating that the property values of the blends need to within the given specifications before it can be fed into a CDU.
- Flowrate limitations per operation type, indicating how much crude can, or must in case of a lower bound, be transferred per day.
- Number of distillation operations, indicating the number of charging operations that must be executed during the schedule. This can also be set by a lower bound and/or upper bound.

An example of a COSP instance is given in Figure 5, which shows the network, and Table 1, which shows the input data. Note that volumes are given in Mbbl, which stands for a 1000 barrels. In this instance there are thus 2 crude vessels, one arriving at the start of the schedule and one on day 4. We can also see that vessel 1 brings 1000 Mbbl of crude type A. For the storage and charging tanks we can see what their capacity is. In this case they all have a minimum capacity of 0 and a maximum capacity of 1000 Mbbl. The list of crude types shows their property values and gross margins respectively. Then there is the list of demanded crude blends, which shows their property specifications and demand of, in this case, 1000 Mbbl respectively. Note that we need to know what crude type is initially in each resource at the start of the schedule, because we need to know their property values. During the schedule these initial crude types can be blended to get crude oil blends, which will have their own property values. The crude oil (blend) in the charging tank should be within the given property specification before it can be fed into a CDU. Finally the flowrate limitations and number of distillations are given.

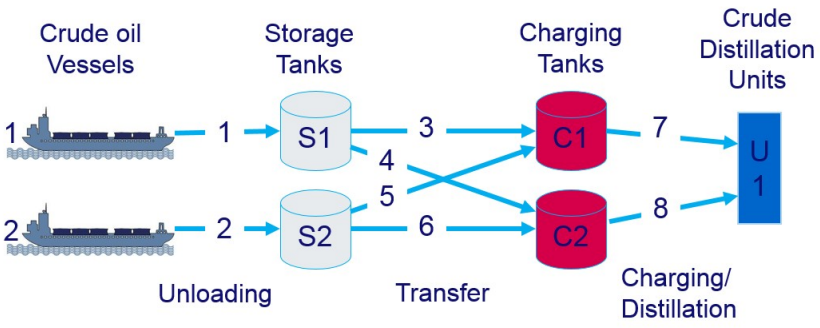

Figure 5: A COSP network example

Table 1: A COSP input data example

Scheduling horizon			8 days
Vessels	Arrival time	Composition	Amount of crude (Mbbl)
Vessel 1	0	100% A	1,000
Vessel 2	4	100% B	1,000
Storage tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 1000]	100% A	250
Tank 2	[0, 1000]	100% B	750
Charging tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]	100% C	500
Tank 2 (mix Y)	[0, 1000]	100% D	500
Crudes	Property 1 (sulfur concentration)		Gross margin ($/bbl)
Crude A	0.01		1
Crude B	0.06		6
Crude C	0.02		2
Crude D	0.05		5
Crude mixtures	Property 1 (sulfur concentration)		Demand (Mbbl)
Crude mix X	[0.015, 0.025]		[1000, 1000]
Crude mix Y	[0.045, 0.055]		[1000, 1000]
Unloading flowrate	[0, 500]	Transfer flowrate	[0, 500]
Distillation flowrate	[50, 500]	Number of distillations	3

2.1.1 *Solution*

The main decisions in the Crude Oil Operations Scheduling Problem are the following:

- decide how many times an operation will be executed before the horizon
- decide on the start of each operation execution
- decide on the duration of each operation execution
- decide on the volume transferred during each operation execution

From these decisions we define a solution to COSP as follows.

We have a set of operations O, which is given by the network, see for example Figure 5 where the arrows represent the operations. We start by defining for each operation $o \in O$ a set of tasks T_o, where the size of T_o, which expresses how often operation o is executed, is defined by $\# : O \to \mathbb{N}$. Let $\mathcal{T}$ be the set of all tasks, i. e., $\mathcal{T} = \bigcup_{o \in O} T_o$. A solution to the COSP is then defined as a tuple $< s, d, v >$ where:

- $s : \mathcal{T} \to \mathbb{R}$ gives the start of the task

- $d : \mathcal{T} \to \mathbb{R}$ gives the duration of the task
- $v : \mathcal{T} \to \mathbb{R}$ gives the volume transferred during the task

The problem is then to find a solution that satisfies all constraints of Section 2.1.3 and maximizes the objective function described in Section 2.1.2. An example of a solution is given in Figure 6. The example shows the operations and when they are executed. Note that the 8 operations are linked to the operations (arrows) seen in the example network, see Figure 5. The arrows indicate which resource is the source and which resource is the target. A volume is transferred from the source to the target. So operation Unloading 1 unloads volume from vessel $V1$ to storage tank $S1$, operation Transfer 3 transfers volume from storage tank $S1$ to charging tank $C1$, operation Distillation 7 charges volume from charging tank $C1$ to CDU $U1$ and so forth. Note that the network data is given as input, meaning that we cannot decide on which operation there are to choose from for the schedule. We cannot for instance add an operation from vessel $V1$ to storage tank $S2$. Also note that vessels will have only one operation, we assume this is because the decision to which storage tank the vessel will unload has already been made.

The volumes can be derived from the tank level changes. For instance, the first unloading operation, Unloading 1, starts after around 1.5 days, has a duration of 2 days, and transfers 1000Mbbl of crude oil, which leads to an increase of the level, by 1000Mbbl crude oil, in storage tank 1.

We can also see that the distillation operations are only executed when their respective charging tank has a crude oil (blend) that is within the given property specifications. If we for instance look at operation Distillation 8, we see a transfer of a 1000Mbbl, which consists of the initial 500Mbbl of crude type D and of the 500Mbbl of crude type B, which was transferred by operation Transfer 6. The property value of crude type D is 0.05 and for crude type B its 0.06. The property value of the crude blend can be calculated using the volumetric average. This will give the crude blend a property value of $(500 * 0.05 + 500 * 0.06)/1000 = 0.055$. The property specification for charging tank $C2$ is $[0.045, 0.055]$, thus the crude blend with a property value of 0.055 is within the property specification.

2.1.2 *Objective Function*

The objective of the optimizer is to maximize the *total gross margin*. The total gross margin is calculated by the volume of crude oil that is fed into a CDU multiplied by their respective gross margin.

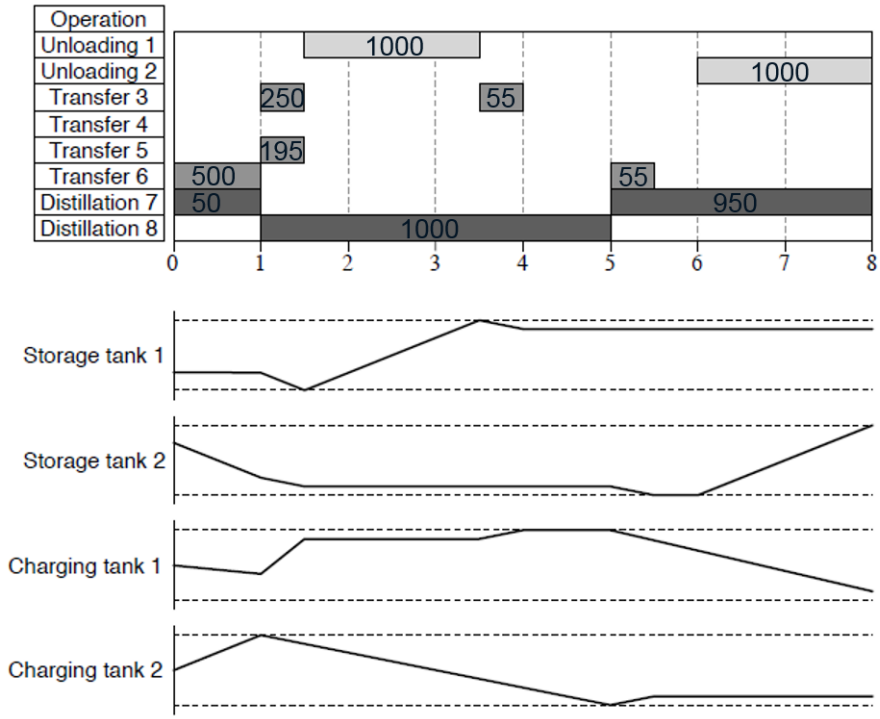

Figure 6: Example of a solution to the COSP

Using the solution definition, the objective function can be formalized as follows. Let $\mathcal{T}^{CH}$ be the set of tasks of distillation operations. The total gross margin is then expressed by:

$$totalgrossmargin = \sum_{t \in \mathcal{T}^{CH}} v(t) \cdot grossmargin_t$$

Where $grossmargin_t$ is the gross margin of the blend of crude oil transferred by t. Figure 7 shows an example of how $grossmargin_t$ is calculated. In the example there is a task t which charges a volume of 550 Mbbl. This volume consists of 250 Mbbl of crude A and 300 Mbbl of crude B. The gross margins of these crude oils are 1 and 3 respectively. $grossmargin_t$ is then calculated by taking the volumetric average which gives us a gross margin of 2.09.

$v(t)$ = 550
$Crude\ A\ volume$ = 250
$Crude\ B\ volume$ = 300

Crude A $grossmargin$ = 1
Crude B $grossmargin$ = 3

Volumetric average:
1 * 250 + 3 * 300 = $grossmargin_t$ * 550

$grossmargin_t$ = 2.09

Figure 7: An example of how the gross margin is calculated

2.1.3 *Constraints*

In this section the different constraints, grouped by subject, are described.

Logistic

The logistics constraints that a solution to the COSP needs to satisfy are the following.

1. Every vessel type (e. g. VLCC) has one unloading berth.
2. Simultaneous inlet and outlet transfers on tanks are forbidden.
3. A tank may charge only one CDU at a time.
4. A CDU can be charged by only one tank at a time.
5. CDUs must be operated continuously throughout the scheduling horizon.

(1) The first logistics constraint allows for different types of vessels and more than one unloading berth, while still keeping the problem as simple as possible. It is possible to make the assumption that a single vessel type has several unloading berths but this overly complicates the problem.

(2) The second logistics constraint only has effect on the storage and charging tanks as the vessels and CDUs only have an outlet or inlet respectively. In some refineries there are waiting constraints on the storage tanks to let brine settle. As the charging tanks are often used to blend crude oil from different storage tanks it is logical to first let the different crude oils blend before the blend is charged into a CDU. Due to this second logistics constraint, inventory tracking in the tanks can be done by looking at the start of the inlet and outlet operations at the tanks instead of looking at the entire interval of the operation.

(3) The third logistics constraint prevents the blend of crude oil to split, as this could change certain properties of the crude oil outside of their demanded specifications.

(4) The fourth logistics constraint prevents the blending of crude oils inside the CDU, which would change the blends created in the charging tanks.

(5) The fifth logistics constraint is a direct refinery constraint as the Component Production Process is a continuous process, which means the process needs a constant input.

Besides the logistics constraints there are also constraints for the tank levels, flowrates, crude blends, compositions, unloading, and demands.

Tank Level

Every tank (storage tanks and charging tanks) may have a lower and/or upper bound in which the level variable needs to lie. Furthermore, the level variable needs to be equal to the initial level plus the volume of inflow operations minus the volume of outflow operations. There is also the constraint which links the individual crude levels to the total level, which is simply that the sum of individual crude levels is equal to the total level.

Flowrate

The flowrate also needs to be within its bounds. The flowrate is calculated using the volume and duration. The volume will be constrained between the flowrate lower bound multiplied by the duration and the flowrate upper bound multiplied by the duration.

Crude Blend

The crude blends that flow into a CDU need to be within certain property bounds. The most important properties, which are sulphur and gravity, have a linear blending behaviour. For these it is easy to express a constraint, but there are also properties that blend in a non-linear way. These latter properties are often either linearized or ignored. In this thesis we will assume that we only have linear properties as input for the COSP.

Composition

The composition constraint expresses that the ratio of crude oils in an outflow operation is equal to the ratio that is in the tank. This is a non-linear constraint (which will be explained in more detail later in Section 4.2), which makes it a constraint that complicates the problem. Therefore, it is important to know that this constraint is not needed when there is only one crude in the tank or when the outflow empties the tank. If these two options are added as constraints then the non-linear constraint disappears, but the final solution might be somewhat too constrained and thus not optimal, see Section 3.2.1 for more detail.

Unloading

Unloading constraints are about the unloading of the crude oil vessels, making sure that everything is unloaded and that the unloading of a vessel does not start until it has arrived.

Demand

Demand constraints are present if there is a demand given. This could be a total demand or a demand per crude blend or even per individual crude.

3 SOLUTION METHODS FOR THE COSP

This chapter will give an overview of the solution methods we implemented to solve the COSP.

Section 3.1 presents the solution method proposed by Mouret 2010 [15]. Section 3.2 presents possible ways to work with the non-linear composition constraint. Section 3.3 presents which approaches were chosen to be implemented in this thesis.

3.1 MOURET'S SOLUTION

In Mouret 2010 [15] a two step decomposition strategy is described to solve the Crude Oil Operations Scheduling Problem (COSP). See Figure 8 for an overview of that strategy. Mouret describes an MINLP model, which can provide an optimal solution but has low performance, meaning that the model can be solved but the solvers will give sub-optimal solutions if not given enough time.

The strategy to improve the performance is to first solve a linearly relaxed MINLP which results in a MIP. The MIP will be used to determine which operations will be used in the solution, how many times an operation will be executed, and in what sequence they will be executed. These results can then be used to fix the sequence of operation executions, which changes the MINLP into an NLP which is easier to solve. Although the MIP does not necessarily give a feasible solution to the COSP, as the non-linear constraints are removed, it does give an upper bound to the problem. The reason is because the MINLP is further constraint with the composition constraint thus will never find a better solution than the MIP. It is important to note that the NLP was solved using a local NLP solver (i. e., CONOPT [1]), thus no proven optimal solution is found.

3.1.1 *Multi Operation Sequencing*

This section will give a summary of *Multi Operation Sequencing* (MOS), which is the time representation used in Mouret 2010 when solving the COSP.

Mouret states that MOS is a time representation that works well when a problem has the following characteristics:

- There is a set of operations that may be executed once, several times, or not executed at all.

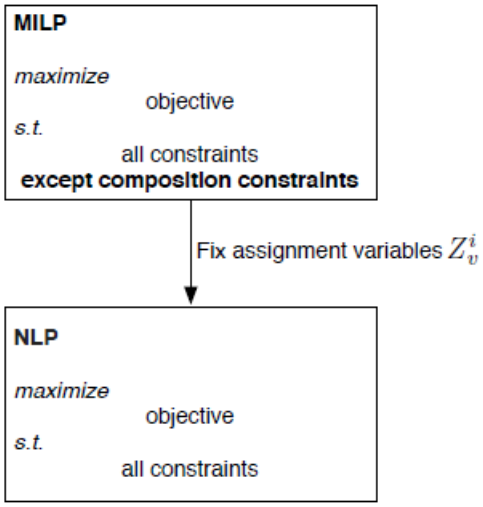

Figure 8: Two step decomposition strategy from Mouret 2010 [15]

- Decisions need to be made on which operations will be executed, when they start, and their duration.
- There are certain scheduling constraints such as due dates and non overlap.
- There are side constraints specific to the problem such as inventory balancing of resources.

MOS was chosen as time representation for the COSP as that problem has these characteristics.

MOS uses an *Overlap Matrix* (*OM*) to extract useful information about the operations. The rows and columns in the overlap matrix correspond to the operations. Every cell in the matrix is thus a combination of two operations. The cells are filled with a 1 or 0, where 1 means that this pair of operations may overlap in time and 0 means that they cannot overlap in time. Let for example v_1 be an operation that transfers crude oil to storage tank s_1 and let v_2 be an operation that transfers crude oil from s_1. If s_1 cannot have an inflow operation and an outflow operation at the same time then $OM_{v_1 v_2} = 0$, which is always the case for storage tanks in the COSP.

Besides the overlap matrix, MOS also makes use of a totally ordered set of *priority slots*, generally referred to as slots $T = 1, ..., n$. These priority slots are used to assign and order the execution of operations. They also provide the possibility for an operation to be executed up to n times (one execution per priority slot), as long as the constraints allow it. The number of priority slots n needs to be set before solving the COSP. A larger n gives a better chance of obtaining the optimal solution but also increases the search space. How n is set and the COSP problem is solved will be explained in Section 4.1.7.

The possible assignments are all the pairs of an operation and a priority slot, where an assignment means that an operation will be executed for a priority slot. Thus if there are six operations and four priority slots, then there are 24 possible assignments. This also means that a single operation can be executed at most four times during a schedule. The following two constraints create a partial scheduling order.

- operations may not be assigned to the same priority slot if they cannot overlap, which is indicated by the OM.
- an operation v_1 is assigned to slot i, and another operation v_2 is assigned to slot j where $i < j$, and the two operations may not overlap, then operation v_1 needs to end before operation v_2 starts.

An example will now illustrate the use of MOS. The example is defined by the resource requirements shown in Table 2 and the OM shown in Table 3. The resource requirements table shows on which resources an operation must be scheduled. Figure 9 presents a possible schedule of this example using MOS. The numbers inside a scheduled task indicate the assigned priority slot. We see that operation v_1 is scheduled twice but in separate priority slots, thus showing that operations can be executed more than once (as long as there is more than one priority slot and no constraints are violated). The schedule also shows that priority slot 1 has two operations that may overlap according to the OM, but that operations v_2 and v_5 are assigned to priority slot 2 as they cannot overlap with operation v_6.

It is important to note that v_2 can overlap with v_3, meaning that there is no precedence constraint between them, but not with v_4. So if for instance v_2 has a longer duration that ends at the start of operation v_4 then v_2 overlaps with v_3 without violating any constraints. This shows that priority slots can overlap if the OM allows it.

Table 2: Resource requirements for the MOS example [15]

Operation	v_1	v_2	v_3	v_4	v_5	v_6
Resources	r_1	r_2	r_3	$r_1 \wedge r_2$	$r_1 \wedge r_3$	$r_2 \wedge r_3$

Table 3: Overlap matrix for the MOS example [15]

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	1	0	0	1
v_2	1	0	1	0	1	0
v_3	1	1	0	1	0	0
v_4	0	0	1	0	0	0
v_5	0	1	0	0	0	0
v_6	1	0	0	0	0	0

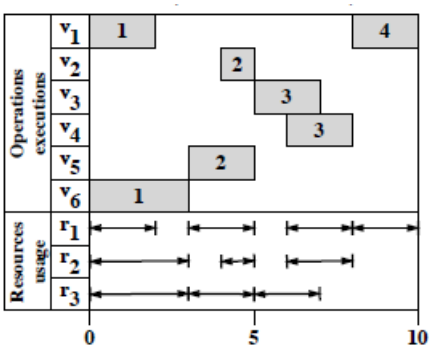

Figure 9: An illustration of a continuous time schedule using MOS [15]

The complete mathematical model can be found in Mouret 2010 [15]. This model also contains the basic constraints used for cardinality constraints, assignment constraints, time constraints, variable bound constraints, non overlapping constraints, and precedence constraints. Several of these constraints are used in the later solution models, see Chapter 4, for the COSP.

3.2 NON-LINEAR SOLUTION

There were several possible solutions found to solve the non-linear problem, divided into three categories: *Avoidance*, *Relaxation*, *Non-linear solver*.

3.2.1 *Avoidance*

Avoiding the non-linear composition constraint means to get a good and usable result without the use of the non-linear composition constraint. This means that the cases for which the non-linear composition constraint is needed are avoided with the help of newly added linear constraints.

For instance, the composition constraint can be avoided by the use of a simple constraint, which will be explained here. The composition constraint makes sure that, when there is a blend in a tank, the crude

ratio in a tank is the same crude ratio that flows out of the tank. There are two cases where the ratios are guaranteed equal without the use of the composition constraint. The first case is that there is only one crude in the tank making both ratios 1 : 1. The second case is that there is a blend of different crude oils but the tank is completely emptied during the outflow, essentially moving the entire blend and thus keeping the ratios equal.

Thus to avoid the composition constraint, one of the cases need to hold for every outflow from a tank. The first case does not need a constraint as it already holds. The second case does need a new constraint to enforce it. This constraint says that the total outflow volume needs to be equal to the total tank level when there is a crude blend in a tank.

Although this is a solution method to solve the non-linearity of the problem, there are a few down sides. The first point is that there might not be a solution at all, for instance if all tanks have blends and there is no possible outflow that can move the entire inventory, because the receiving tank does not have enough capacity. Another example is that there might be minimum capacity constraints that prohibits the transfer of the entire inventory. Second point is that the solution, although feasible and usable, is not always optimal and may not even be close to being optimal. For instance, it could be that it is far more profitable to not empty a tank in one outflow operations because the remaining blend can be further blended with a high value crude oil and get the blend within specifications.

It is important to note that a solution found by this approach is a feasible solution to the COSP.

3.2.2 *Relaxation*

Another way to solve a non-linear problem is by linear relaxation of the non-linear constraints. One way of doing this is by using the McCormick envelope and McCormick cuts [12] [15]. How this method works will be explained by an example. Let the non-linear constraint be the following: $X_{iv} = A_{iv}V_{iv}$, where X, A, and V are continuous variables and i and v are indices for priority slots and operations respectively. For every variable a lower and upper bound is known, for instance variable X has lower bound X^L_{iv} and upper bound X^U_{iv}.

Note that the non-linear composition constraint is a bit different, looking more like the following: $L_{iv}W_{iv} = A_{iv}V_{iv}$, which in its turn can be written as $L_{iv}W_{iv} = X_{iv} = A_{iv}V_{iv}$. Thus the non-linear composition constraint is twice as big as the example. The detailed non-linear composition constraint can be found in Section 4.2.

Next step is the linear relaxation of the non-linear constraint. The non-linear constraint is replaced by the following constraints.

$$
\begin{aligned}
X_{iv} &\geq A^L_{iv} V_{iv} + A_{iv} V^L_{iv} - A^L_{iv} V^L_{iv} \\
X_{iv} &\geq A^U_{iv} V_{iv} + A_{iv} V^U_{iv} - A^U_{iv} V^U_{iv} \\
X_{iv} &\leq A^L_{iv} V_{iv} + A_{iv} V^U_{iv} - A^L_{iv} V^U_{iv} \\
X_{iv} &\leq A^U_{iv} V_{iv} + A_{iv} V^L_{iv} - A^U_{iv} V^L_{iv}
\end{aligned}
$$

Figure 10 will be used to illustrate why these constraints can replace the non-linear constraint. The figure shows a graph where the horizontal axis is used for variable V and the vertical axis is used for variable A. Note that variable X is defined as the area of $A \cdot V$. The light blue area illustrates the right hand side of the first constraint and the red box illustrates the area of X. As shown the area of X is larger than the light blue area. We can also immediately see that the blue area cannot be larger than the area of X. We also see that the closer the lower bounds get to the chosen values of A and V, the more area is covered by the light blue area within the red box. Using this illustration and deriving the other constraints in the same way we see that these constraints can indeed replace the non-linear constraint as long as the bounds can get close enough to the values of A and V.

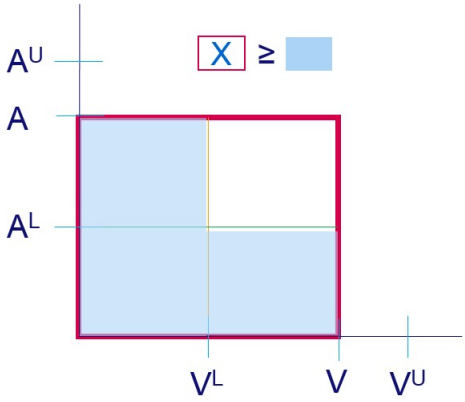

Figure 10: Illustration of the first constraint

Now that the non-linear constraint is replaced by four linear constraints the model has become linear. But it will not find the right solution without a way to set the lower and upper bounds. This is done in an iterative way where the bounds are gradually contracted until their difference is within a set error value. To contract a bound the method uses McCormick cuts. In general a cut is a constraint that invalidates the current infeasible solution but does not remove any feasible solutions. When a feasible solution is found the bounds can be contracted using constraint propagation. This method might be implement in Quintiq but is too complex for this thesis. Figure 11 shows an overview of the method to contract the bounds. There are

also slightly different methods to contract the bounds such as shown in Karuppiah et al. 2007 [9].

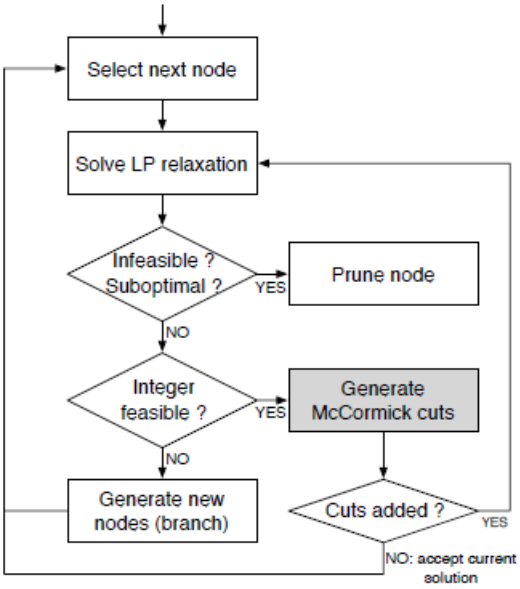

Figure 11: Branch and cut algorithm with McCormick cuts [15]

3.2.3 *Non-linear solver*

The last option is to use a solver that can handle non-linear constraints.

The first possibility is making use of constraint programming, which is integrated in Quintiq's software. The second possibility is a commercial tool called LocalSolver [3]. The third possibility is a commercial NLP solver called CONOPT [1]. CONOPT was used in Mouret 2010 [15] as the solver for the second step where it showed good solutions and performance.

3.3 IMPLEMENTED SOLUTIONS

As project time is limited some choices had to be made on which possible solutions to chase. The first choice that was implemented was the two-step approach of Mouret where the second step uses CP instead of an NLP solver. The second choice was the approach that avoids the non-linear composition constraint, called 1-Step Heuristic. The third choice was to look at LocalSolver to replace the CP as the second step in the two-step approach of Mouret. The fourth choice was to look at CONOPT to be the NLP solver for the second step. Lastly, we also looked into other possible heuristic approaches, which will be discussed at a later stage. The implemented methods are further explained in Chapter 4

4

SOLUTION MODELS FOR THE COSP

This chapter presents the best solution method for the COSP known in literature, being Mouret's MIP model, in Section 4.1. Section 4.2 presents a Constraint Programming model, followed by a MIP-based heuristic in Section 4.3, and another MIP-based heuristic in Section 4.4. The LocalSolver model is given in Section 4.5, and we end the chapter with a discussion on performance in Section 4.6, where we look into several possibilities to improve Mouret's MIP and the CP model.

4.1 MOURET'S MIP

This section gives a mathematical model of the COSP. The definition consists of sets, parameters, variables, constraints and an objective function. This initial model is taken from Mouret 2010 [15].

4.1.1 *Sets*

The model has the following sets to represent the resources, operations, and crude oils with their properties.

- $T = 1, ..., n$ is the set of priority slots
- R_V is the set of vessels
- R_S is the set of storage tanks
- R_C is the set of charging tanks
- R_D is the set of CDUs
- R is the set of resources: $R = R_V \cup R_S \cup R_C \cup R_D$
- W_U is the set of unloading operations from vessels to storage tanks
- W_T is the set of transfer operations from storage to charging tanks
- W_D is the set of charging operations from charging tanks to CDUs
- W is the set of operations: $W = W_U \cup W_T \cup W_D$
- I_r is the set of inlet operations on resource r
- O_r is the set of outlet operations on resource r

- C is the set of crude oils
- K is the set of crude oil properties (e. g. sulfur concentration)

Note the sets of resources and operations are defined by the network of an instance, such as seen in Figure 5. While the sets of crude oils and properties are defined by the input data of the instance, such as seen in Table 1. Finally the set of priority slots is defined by an algorithm that is later explained in this section.

4.1.2 *Input Data*

The model has the following input data.

- H is the scheduling horizon
- $Type_r$ is the vessel type of vessel $r \in R_V$
- $[\underline{V_v^t}, \overline{V_v^t}]$ are bounds on the total volume transferred during operation $v \in W$. Note that the superscript t simply denotes that it is the *total* volume. In all instances, $\underline{V_v^t} = 0$ except for unloading where it holds that $\underline{V_v^t} = \overline{V_v^t}$ being the volume of crude in the crude vessel
- $[\underline{N_D}, \overline{N_D}]$ are the bounds on the number of distillations
- $[\underline{FR_v}, \overline{FR_v}]$ are the bounds on the flowrate for operation $v \in W$
- $\underline{S_v}$ is the earliest possible start time of unloading operation $v \in W_{UI}$ (i. e., arrival time of the corresponding crude vessel)
- $[\underline{x_{vk}}, \overline{x_{vk}}]$ are the bounds of property k of the blended products transferred during operation $v \in W_D$. The input data links the property specification to the charging tank, thus these operation property specification comes from its source resource, which is generally a charging tank.
- x_{ck} is the value of property k in crude c
- $[\underline{L_r^t}, \overline{L_r^t}]$ are the *total* capacity bounds of tank r
- L_{0r}^t is the initial *total* level inside tank r
- L_{0rc} is the initial crude level inside tank r for crude c
- $[\underline{D_r}, \overline{D_r}]$ are the bounds of the demand on products to be transferred out of the charging tank $r \in R_C$ during the scheduling horizon. Thus the sum of the volumes of all outflow operations of r needs to be within the demand bounds.
- G_c is the gross margin of crude c

4.1.3 *Variables*

The variables used in the model are composed of binary assignment variables, and continuous time, operation and resource variables.

- *Assignment variables* $Z_{iv} \in \{0,1\} \qquad i \in T, v \in W$

$$Z_{iv} = \begin{cases} 1, & \text{if operation } v \text{ is assigned to priority-slot } i \\ 0, & \text{otherwise} \end{cases}$$

- *Time variables* $S_{iv} \geq 0, D_{iv} \geq 0, E_{iv} \geq 0 \qquad i \in T, v \in W$
  S_{iv} is the start time of operation v if it is assigned to priority-slot i, $S_{iv} = 0$ otherwise.
  D_{iv} is the duration of operation v if it is assigned to priority-slot i, $D_{iv} = 0$ otherwise.
  E_{iv} is the end time of operation v if it is assigned to priority-slot i, $E_{iv} = 0$ otherwise.

- *Volume variables* $V^t_{iv} \geq 0$ and $V_{ivc} \geq 0 \qquad i \in T, v \in W, c \in C$
  V^t_{iv} is the *total* volume of crude transferred during operation v if it is assigned to priority-slot i, $V^t_{iv} = 0$ otherwise.
  V_{ivc} is the volume of crude c transferred during operation v if it is assigned to priority-slot i, $V_{ivc} = 0$ otherwise.

- *Resource variables* L^t_{ir} and $L_{irc} \qquad i \in T, r \in R, c \in C$
  L^t_{ir} is the *total* accumulated level of crude in tank $r \in R_S \cup R_C$ before the operations assigned to priority-slot i.
  L_{irc} is the accumulated level of crude c in tank $r \in R_S \cup R_C$ before the operations assigned to priority-slot i.

The decision variables are Z_{iv}, S_{iv}, D_{iv}, and V_{ivc}. These decision variables can be related to the solution definition, see Section 2.1.1, in the following way. Every assignment, meaning $Z_{iv} = 1$, is a task where v is the operation the task performs. The size of T_o (the function #), where $o = v$, can then be derived from the number of assignments Z_{iv} for a certain operation v. Every assignment can be linked to a task t, where $s(t)$ and $d(t)$ can be directly set from S_{iv}, D_{iv}. $v(t)$ is the sum of V_{ivc} for the different crude oils, which is the same as variable V^t_{iv}.

4.1.4 *Objective Function*

The objective is to maximize the summed total of gross margins of the distilled crude blends, which represents the approximate revenue of the crude blends. Using the individual gross margins G_c it is calculated as follows.

$$\textit{maximize} \sum_{i \in T} \sum_{r \in R_D} \sum_{v \in I_r} \sum_{c \in C} G_c \cdot V_{ivc}$$

4.1.5 *Constraints*

This section contains all the constraints in Mouret's MIP model.

Time constraints

Constraint 4.1a says that a vessel cannot unload before it has arrived. Constraint 4.1b makes sure that all assigned operations end before the end of the time horizon. Constraint 4.1c expresses that start plus duration is equal to the end of an execution of an operation.

$$S_{iv} \geq \underline{S_v} \cdot Z_{iv} \qquad i \in T, v \in W_U \tag{4.1a}$$

$$E_{iv} \leq H \cdot Z_{iv} \qquad i \in T, v \in W \tag{4.1b}$$

$$E_{iv} = S_{iv} + D_{iv} \qquad i \in T, v \in W \tag{4.1c}$$

Distillation constraints

The distillation constraint 4.2 sets a bound on the number of charging operations. This way the model has control on the number of switches which is important as it costs a lot to switch charging operations.

$$\underline{N_D} \leq \sum_{i \in T} \sum_{v \in W_D} Z_{iv} \leq \overline{N_D} \tag{4.2}$$

Overlap constraints

Constraint 4.3 says that two different operations that may not overlap may not be assigned to the same priority slot. OM represents the overlap matrix between all the operations and gives 0 if the operations may not overlap, see Section 3.1.1. $v_1 < v_2$ is added instead of $v_1 \neq v_2$ to reduce the number of constraints, because $OM_{v_1 v_2} = OM_{v_2 v_1}$. To know which operation is "smaller", id's were added to the operations.

$$Z_{iv_1} + Z_{iv_2} \leq 1 \qquad i \in T, v_1, v_2 \in W, v_1 < v_2, OM_{v_1 v_2} = 0 \tag{4.3}$$

When two operations may not overlap then a precedence is set between the two. Constraint 4.4 looks at one operation and expresses that the operation should end before the start of the same operations that are assigned to later priority slots. Constraint 4.5 looks at two different operations and expresses that the operation assigned to a lower priority slot should end before the start of an operation that is assigned to a higher priority slot. Note that of the pairs of start, end and assignment variables one will always be zero, because of the time constraints and the first overlap constraint.

$$E_{i_1 v_1} \leq S_{i_2 v_1} + H \cdot (1 - Z_{i_2 v_1}) \qquad i_1, i_2 \in T, i_1 < i_2, v_1 \in W, OM_{v_1 v_1} = 0 \tag{4.4}$$

$$E_{i_1 v_1} + E_{i_1 v_2} \leq S_{i_2 v_1} + S_{i_2 v_2} + H \cdot (1 - Z_{i_2 v_1} - Z_{i_2 v_2}) \qquad i_1, i_2 \in T, i_1 < i_2, v_1, v_2 \in W, v_1 \neq v_2, OM_{v_1 v_2} = 0 \tag{4.5}$$

Crude vessel constraints

These constraints set the unloading precedence on the vessel unloading, ships will be unloaded in the sequence they arrive. Constraint 4.6 expresses a vessel should be unloaded before a later arriving vessel of the same vessel type starts unloading, whereas constraint 4.7 expresses that the unloading operation of a vessel needs to be assigned to a lower slot than the unloading operation of a later arriving vessel of the same vessel type.

$$\sum_{i \in T} \sum_{v \in O_{r_1}} E_{iv} \leq \sum_{i \in T} \sum_{v \in O_{r_2}} S_{iv} \qquad r_1, r_2 \in R_V, r_1 < r_2, Type_{r_1} = Type_{r_2} \tag{4.6}$$

$$\sum_{\substack{j \in T \\ j<i}} \sum_{v \in O_{r_1}} Z_{jv} \leq \sum_{\substack{j \in T \\ j \leq i}} \sum_{v \in O_{r_2}} Z_{jv} \qquad i \in T, r_1, r_2 \in R_V, r_1 < r_2, Type_{r_1} = Type_{r_2} \tag{4.7}$$

Constraint 4.8 is a cardinality constraint that expresses that vessels must be unloaded in one unload operation.

$$\sum_{i \in T} \sum_{v \in O_r} Z_{iv} = 1 \qquad r \in R_V \tag{4.8}$$

CDU constraints

constraint 4.9 expresses that CDUs need to be charged continuously during the entire horizon.

$$\sum_{i \in T} \sum_{v \in I_r} D_{iv} = H \qquad r \in R_D \tag{4.9}$$

Volume constraints

Constraint 4.10a expresses that when an operation is assigned its volume should be under its upper bound. The upper bound can be set to the maximal capacity of the resource the flow originates from. For unloading operations it can be set to the initial amount.
Constraint 4.10b expresses that when an operation is assigned its volume should be above its lower bound. The lower bound can be set to the initial amount for unloading operations, and to zero for the rest.
Constraint 4.10c expresses that he volumes of the different crude oils should be equal to the total volume transferred.

$$V^t_{iv} \leq \overline{V^t_v} \cdot Z_{iv} \qquad i \in T, v \in W \tag{4.10a}$$

$$V^t_{iv} \geq \underline{V^t_v} \cdot Z_{iv} \qquad i \in T, v \in W \tag{4.10b}$$

$$V^t_{iv} = \sum_{c \in C} V_{ivc} \qquad i \in T, v \in W \tag{4.10c}$$

Level constraints

Constraint 4.11a expresses that the level in a tank at the start of a priority slot is equal to initial level plus all the volume transferred in prior minus the volume transferred out prior.
Constraint 4.11b expresses the same for the individual crude levels inside a tank.
Constraint 4.11c expresses that the sum of all the individual crude oil levels in a tank should add up to the total level inside a tank.

$$L^t_{ir} = L^t_{0r} + \sum_{\substack{j \in T \\ j<i}} \sum_{v \in I_r} V^t_{iv} - \sum_{\substack{j \in T \\ j<i}} \sum_{v \in O_r} V^t_{iv} \qquad i \in T, r \in R \tag{4.11a}$$

$$L_{irc} = L_{0rc} + \sum_{\substack{j \in T \\ j<i}} \sum_{v \in I_r} V_{ivc} - \sum_{\substack{j \in T \\ j<i}} \sum_{v \in O_r} V_{ivc} \qquad i \in T, r \in R, c \in C \tag{4.11b}$$

$$L^t_{ir} = \sum_{c \in C} L_{irc} \qquad i \in T, r \in R \tag{4.11c}$$

Operation constraints

Constraint 4.12a expresses that the flowrate of an operation should be within its bounds. Constraint 4.12b expresses that the crude (blend) that is charged during a charging operation should be within the given property specification. Note that the operation property specifications come from its source which is a charging tank. Note that we assume that the properties blend in a linear way.

$$\underline{FR_v} \cdot D_{iv} \leq V^t_{iv} \leq \overline{FR_v} \cdot D_{iv} \qquad i \in T, v \in W \tag{4.12a}$$

$$\underline{x_{vk}} \cdot V^t_{iv} \leq \sum_{c \in C} x_{ck} V_{ivc} \leq \overline{x_{vk}} \cdot V^t_{iv} \qquad i \in T, v \in W, k \in K \tag{4.12b}$$

Capacity constraints

Constraint 4.13a expresses that the total level inside a tank should be between its bounds during the scheduling horizon. Constraint 4.13b expresses that the level of a single crude inside a tank should be between zero and the upper bound of the tank. constraints 4.13c and 4.13d express that the total level and individual crude levels variables are also within bounds at the end of the horizon.

$$\underline{L^t_r} \leq L^t_{ir} \leq \overline{L^t_r} \qquad i \in T, r \in R_S \cup R_C \tag{4.13a}$$

$$0 \leq L_{irc} \leq \overline{L^t_r} \qquad i \in T, r \in R_S \cup R_C, c \in C \tag{4.13b}$$

$$\underline{L^t_r} \leq L^t_{0r} + \sum_{i \in T} \sum_{v \in I_r} V^t_{iv} - \sum_{i \in T} \sum_{v \in O_r} V^t_{iv} \leq \overline{L^t_r} \qquad r \in R_S \cup R_C \tag{4.13c}$$

$$0 \leq L_{0rc} + \sum_{i \in T} \sum_{v \in I_r} V_{ivc} - \sum_{j \in T} \sum_{v \in O_r} V_{ivc} \leq \overline{L^t_r} \qquad r \in R_S \cup R_C, c \in C \tag{4.13d}$$

Demand constraints

Constraint 4.14 expresses that the total outflow from a charging tank is within the bounds of its demand.

$$\underline{D_r} \leq \sum_{i \in T} \sum_{v \in O_r} V_{iv}^t \leq \overline{D_r} \qquad r \in R_C \tag{4.14}$$

4.1.6 *Additional Constraints*

This section presents constraints that were added to the core constraints, by Mouret, to enhance the performance of the model.

4.1.6.1 *Symmetry constraints*

There is symmetry in the model, which means that two different solutions are equivalent and have the exact same objective value. An operation could for instance be assigned to priority slot one or priority slot two without changing the rest of the solution. This degrates performance because the solver can be wasting time finding a solution that is in essence already found. To remedy this, constraint 4.15 was added to the model. The constraint says that an operation cannot be assigned to priority slot i if it can also be assigned to priority slot $i-1$.

$$Z_{iv} \leq \sum_{\substack{v' \in W \\ OM_{vv'}=0}} Z_{(i-1)v'} \qquad i \in T, i > 1, v \in W \tag{4.15}$$

4.1.6.2 *Strengthening constraints*

There is also the possibility of adding strengthening constraints, by creating sets of operations that may not overlap with each other. In this case maximal sets of operations are constructed that may not overlap with each other, where a maximal set needs to have at least 3 operations. A set is maximal when there is no operation left outside the set that cannot overlap with any of the operations already in the set. For every maximal set W' the following constraints are added. Constraint 4.16 says that for every priority slot only one operation in the set may be assigned. Constraint 4.17 says that the end of an operation in the set W', plus the duration of operations in the set assigned to priority slots in between, should be before the start of an operation assigned to a higher slot.

$$\sum_{v \in W'} Z_{iv} \leq 1 \qquad i \in T \tag{4.16}$$

$$\begin{aligned} &\sum_{v \in W'} E_{i_1 v} + \sum_{\substack{i \in T \\ i_1 < i < i_2}} \sum_{v \in W'} D_{iv} \\ &\leq \sum_{v \in W'} S_{i_2 v} + H \cdot (1 - \sum_{v \in W'} Z_{i_2 v}) \end{aligned} \qquad i_1, i_2 \in T, i_1 < i_2 \tag{4.17}$$

4.1.6.3 *Cardinality constraints*

In Mouret 2010 [15] there was nothing mentioned about other cardinality constraints besides the vessel unloading, see Constraint 4.8, and number of distillation bounds, see Constraint 4.2. However we found the model Mouret, used to run his COSP experiments, published online [14]. When we went through the model we saw two extra cardinality constraints.

The first cardinality constraint 4.18 expresses that the number of executions of a certain charging operation needs to be within given bounds. The bounds will be expressed as $[\underline{MinCard_v}, \overline{MaxCard_v}]$, where $v \in W_D$.

$$MinCard_v \leq \sum_{i \in T} Z_{iv} \leq MaxCard_v \qquad v \in W_D \tag{4.18}$$

The second cardinality constraint 4.19 expresses that for the first priority slot and for every CDU there needs to be at least one incoming operation assigned. For this we define the set W_{to-r}, which includes all operations that have a given resource r as target.

$$\sum_{v \in W_{to-r}} Z_{1v} = 1 \qquad r \in R_D \tag{4.19}$$

We will do a comparison between the model with and without these constraints to see how important they are to the performance, as they were left out of the dissertation paper, , see Section 5.3.2 for the results. Note that to do this we need to have values for the bounds in Constraint 4.18. We can take the values used by Mouret for his instances but they differ per instance and we also have new instances. So to get these values we propose a rule to get values for the bounds.

The rule is based on the notion that the number of outflows from the charging tanks is balanced. This happens because of the logistic assumptions that say that a CDU can only have one inflow operation at a time, a charging tank can only have one outflow operation at a time, and a CDU needs to be charged continuously. Together with the given networks we will get a wave effect with the distillation operation executions, whereby the charging tanks will for instance first use their first operation and then at some point need to change to using their second operation. Also note that the first and last charging tanks only have one outgoing operation whereby the other charging tanks will have two outgoing operations. This implies that to keep the balance we need to count the operations for the first and last charging tanks double.

The rule to set the $MinCard_v$ is defined as follows: $nrCH \cdot 2 - 2 + x(nrCH \cdot 2 - 2) \leq \underline{N_D}$, where $nrCH$ denotes the number of charging tanks and x is the integer decision variable. We want to maximize

this x. Then the $MinCard_v$ is set to $x + 1$ for the outlet operations v of a charging tank. For example if $nrCH = 2$ and $\underline{N_D} = 3$, then we get $x = 0$. Thus the $MinCard_v$ are set to 1, which is logical because to balance the minimum of 3 distillation operations both charging tanks need to have at least one outlet operation, while setting the minimum to two would bring it to four outlet operations which is one too many.

The rule to set the $MaxCard_v$ is defined as follows: $nrCH \cdot 2 + x(nrCH \cdot 2 - 2) \geq \overline{N_D}$, where $nrCH$ denotes the number of charging tanks and x is the integer decision variable. We want to minimize this x. Then the $MaxCard_v$ is set to $x + 2$ for the outlet operations v of the first and last charging tank, while the $MaxCard_v$ is set to $x + 1$ for the outlet operations v of the other charging tanks. For example if $nrCH = 2$ and $\overline{N_D} = 3$, then we get $x = 0$. As we only have two charging tanks they are the first and last tanks, thus their outlet operations are set to $MaxCard_v = 2$. This is again logical as having less than 2 would bring the total possible distillations to 2, which is not enough, and adding to this would possibly break up the balance, because one charging tank could then take all the distillation operations.

4.1.7 *Priority Slot Algorithm*

The MIP model in Section 4.1 assumes one has n priority slots. This section will show how to find a good or the best value for n and simultaneously solve the COSP. The method was taken from Mouret 2010 [15]. When an optimal solution is found, it is optimal for the n priority slots but adding a slot might give a better solution, because by adding a priority slot every operation has the possibility of an extra execution, thus allowing for new solutions. The less priority slots there are however, the faster a solution can be found.

The strategy of Mouret 2010 [15] is an additive approach, which starts by solving the COSP using one priority slot and then keep solving the COSP with an added priority slot each iteration until a stopping criterion is reached. There are several possible stopping criteria. A first criterion is to stop when the variation between the previous objective value and the current objective value is smaller than a set error margin. A second criterion is to stop when a set upper bound of priority slots is reached. A third criterion is to set a time limit. Only the second criterion can guarantee global optimality if the upper bound for n is known, though there is no known method to get this upper bound for the COSP. Although the first criterion does not guarantee that global optimality will be reached, experiments performed by Mouret show that global optimality was reached in most cases. Note that Mouret tested this algorithm on more problems than only the COSP.

By adding priority slots, the search space is increased but during the previous search a large part was already searched. To ignore the

previous search space, constraint 4.20 is added, which says that all priority slots need to have at least one assignment. This way the search space only considers new solutions.

$$\sum_{v \in W} Z_{iv} \geq 1 \qquad i \in T \tag{4.20}$$

Another way of increasing performance is by setting the *cutoff value*, which is the minimal allowed objective value, to the objective value of the previous found solution. This means that solutions with smaller objective values than the cutoff are not considered as feasible solutions.

The algorithm that solves the problem and finds the best value for n is given by Algorithm 1. Here z and z^* are the current and previous solutions respectively, n is the number of priority slots, and n_0 is the initial number of priority slots. As the optimizer maximizes the objective, the cutoff is initially set to negative infinity. The optimizer call is indicated by $Maximize(MOS(n), cutoff)$.

Algorithm 1 Additive approach to solving the COSP and finding a good n, Mouret 2010 [15]

```
z*← ∅;
cutoff ← −∞;
n ← n0;
repeat
  z ←Maximize(MOS(n), cutoff);
  Δ ← z.objectivevalue() - z*.objectivevalue();
  if Δ > 0 then
    z*← z;
    cutoff← z*.objectivevalue();
  end if
  n ← n + 1;
until stopping criteria;
return z*;
```

4.2 CONSTRAINT PROGRAMMING MODEL

This section proposes a CP model that implements the second step in Mouret's two step decomposition strategy, where Mouret used an NLP model and a non-linear solver named CONOPT. The CP model will use the solution of Mouret's MIP to get the assigned operations, with their sequence, as additional input for the CP model. This means that the assignment variable is not needed in the CP model, i. e., the CP model does not have to decide on the number of executions per operation and the sequence of all the chosen executions. It still needs to decide on the start, duration, and crude volumes. Thus the other

variables are still needed and are given a range for the CP to work with. The time variables get the range $[0, horizon.inseconds]$. It is important to notice that CP only works with integer decision variables and the horizon, which was first expressed in days, is now converted to seconds. This conversion also holds for the arrival times of the vessels. The level ranges need to be set to $[\underline{L_r^t}, \overline{L_r^t}]$ and the volume ranges are set to $[0, \overline{L_r^t}]$, where resource r is the source resource. The quantities given by the data are in Mbbl (1000 barrels), which may be increased by a quantity factor to allow for more solutions, but which will increase the search space. We do this because we would then allow for more precise volumes to transfer, which could result in higher valued crude blends.

The CP model has the same objective function, defined in Section 4.1.4, as the MIP. It also uses the following constraints from the MIP: 4.1a, 4.1c, 4.9, 4.10a, 4.10b, 4.10c, 4.11a, 4.11b, 4.11c, 4.12a, 4.12b, 4.13a, 4.13b, 4.13c, 4.13d, and 4.14.

The overlap and precedence constraints can be merged into one simple constraint because the sequence of operation executions is already fixed. This constraint looks at the assigned operations and says that operations on lower priority slots should end before the start of operations on higher priority slots if they may not overlap:

$$E_{i_1 v_1} \leq S_{i_2 v_2} \qquad i_1, i_2 \in T, i_1 < i_2, v_1, v_2 \in W, \\ OM_{v_1 v_2} = 0, Z_{i_1 v_1} = 1, Z_{i_2 v_2} = 1$$

The last constraint is the non-linear composition constraint for which this CP was created. This constraint says that the crude ratio flowing out of a tank should be the same as the crude ratio in the tank. We will explain why this constraint is needed with the help of an example given in Figure 12. In the figure we see a storage tank $S1$ and a transfer operation 4. $S1$ holds a volume of 100 Mbbl, where 50 Mbbl is of type crude A, and 50 Mbbl is of type crude B. This means that the crude ratios in $S1$ are $1 : 2$. Now we have a transfer operation 4, which transfers 50 Mbbl. What we want to know is how many of the 50 Mbbl that is transferred is of crude type A and how much is of crude type B. Without the composition constraint this could be anything, for instance the 50 Mbbl that is transferred could consist of only crude type A. But in reality this is physically impossible of course because the crude types in $S1$ are blended, meaning that we cannot choose how much of which crude type we want to transfer. Using the composition constraints this is also taken into account in the model. Because this constraint says that the ratios are equal thus the ratio of the crude types in the transferred volume needs to be $1 : 2$ for both types. As such the 50 Mbbl that is transferred consists of 25 Mbbl of crude type A and 25 Mbbl of crude type B.

The composition constraint looks as follows:

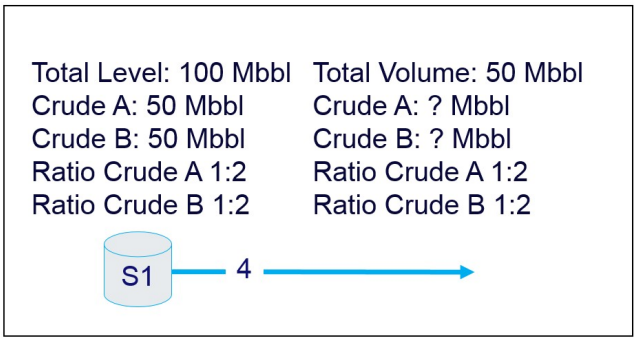

Figure 12: Illustration of the composition constraint

$$V_{ivc} \cdot L^t_{ir} = L_{irc} \cdot V^t_{iv} \qquad i \in T, r \in R, v \in O_r, c \in C$$

The proposed model can be found in Appendix 11 in OPL code.

We also propose a variant on the CP. This variant only partly sets the assignments. Which means that the assignments obtained by the MIP are set to 1, but the other possible assignments are not set to 0, i. e., these assignments can still become 1 if possible. This approach allows for more solutions and possibly better solutions than can be found using the first variant, while still reducing the total search space by a large factor compared to when there are no fixed assignments.

This proposed variant can be found in Appendix 12 in OPL code.

4.2.1 *Search Strategy*

The search strategy that was found to be a good fit for this CP model is *Restart*. Restart is a search strategy that is standard implemented in *IBM ILOG CPLEX CP Optimizer version 12.6*. The precise workings of this strategy are confidential, but we can infer that it is a strategy which starts over when a certain limit is reached. This limit can be time, depth of the search tree, etc.

The search strategy can be further influenced by search phases. These search phases tell the search which set of variables should be focused on first. Here the first search phase is set to the crude volume variables as all the level and volume variables can be calculated from this variable and the objective function is also calculated by this variable. Next is the search phase where the start variables are focused on, which means that the time intervals are moved to a place such that the operations that may not overlap are not overlapping and are in the right sequence. The OPL code of the search phases looks as follows:

$$cp.setSearchPhases(f.searchPhase(cvolume), f.searchPhase(start));$$

4.3 1-STEP HEURISTIC

In Section 3.2.1 a non-linear avoidance approach was described as a possible solution method to avoid the non-linear composition constraint. This section proposes a MIP that implements this approach, called *1-Step Heuristic*. 1-Step Heuristic avoids the non-linear constraint by adding some new additional variables and linear constraints to Mouret's MIP. This model has two variants. The first is using it as described, producing a solution to the entire problem using n priority slots. The second is when Mouret's MIP is run first and the assignments are then locked for the 1-Step Heuristic, thus essentially replacing the second step.

The constraint that needs to be added is the one that sets the outflow of crude of a resource equal to the crude level in the resource if there is a crude blend in the resource. Meaning that if there is a blend of crude oils in a tank, then the tank should be emptied in one operation. To add this constraint two sets of binary variables are added. The first set of binary variables express if the level of a crude c in a resource r in a priority slot i is greater than zero. The binary will be set to 1 if the level is greater than zero, and to 0 otherwise. The second set of binary variables will check if there is a blend in resource r in a priority slot i by counting the number of binary variables set to 1 in the first set of binary variables. The binary will be set to 1 if there is a blend in the resource, and to 0 otherwise.

- *crude level binary* $CL_{irc} \in \{0,1\} \qquad i \in T, r \in R, c \in C$

$$CL_{irc} = \begin{cases} 1, & \text{if } L_{irc} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- *crude blend binary* $CB_{ir} \in \{0,1\} \qquad i \in T, r \in R$

$$CB_{ir} = \begin{cases} 1, & \text{if } \sum_{c \in C} CL_{irc} > 1 \\ 0, & \text{otherwise} \end{cases}$$

The following constraint 4.21 is used to set the crude level binary. Note that L_{irc} cannot be between 0 and 1bbl, because of the way the lower bound of the constraint is set.

$$0.001 * CL_{irc} \leq L_{irc} \leq \overline{L_r^t} \cdot CL_{irc} \qquad i \in T, r \in R, c \in C \tag{4.21}$$

The following constraint 4.22 is used to set the crude blend binary. It is a big-M constraint where the M is equal to the number of crude oils that could have a level in the resource, a method to get these crude oils is later explained in Section 4.6.1.1.

$$2 \cdot CB_{ir} \leq \sum_{c \in C} CL_{irc} \leq M \cdot CB_{ir} + 1 \qquad i \in T, r \in R \tag{4.22}$$

The last constraint 4.23 sets the total volume equal to the total level if the crude blend binary is 1. Note that the total volume is already at most the total level, which means that the constraint only needs to add that the total volume needs to be at least the total level when a crude blend is present. The constraint is a big-M constraint where M is equal to the maximal capacity of the resource. The assignment variable is added so that this constraint only needs to hold when the outflow operation is actually scheduled.

As explained, the problem instances do not always allow for a feasible solution where the tanks with a crude blend can be emptied in one operation. Because of these cases we add a continuous $error_{iv}$ variable, so that it will allow for such a tank to not be emptied completely in one operation but we do want the tank to be as empty as possible.

$$\overline{L_r^t}(1 - Z_{iv}) + \overline{L_r^t}(1 - CB_{ir}) + V_{iv}^t + error_{iv} \geq L_{ir}^t \qquad i \in T, r \in R, v \in O_r \tag{4.23}$$

As such the objective function is changed to incorporate the $error_{iv}$ variable. At this point we rather want the tanks with a crude blend to empty in one operation than a high total gross margin, because we are looking for a feasible solution to the entire problem. Thus the weight for the error will be higher than the weight for the gross margin. The weight is set to 10^5, it needs to be this high to be as independent as possible regarding the total gross margin. Note that if it is the other way around we get the same results as Mouret's MIP. The new objective function 4.24 looks as follows:

$$maximize \sum_{i \in T} \left(\sum_{r \in R_D} \sum_{v \in I_r} \sum_{c \in C} (G_c \cdot V_{ivc}) - \sum_{v \in W} (error_{iv} * 10^5) \right) \tag{4.24}$$

The idea is that emptying a tank with a crude blend as much as possible will keep the transgression of the composition constraint to a minimum. Thus obtaining a result that is an actual solution or is close to an actual solution to the COSP.

4.4 ANOTHER HEURISTIC

This section discusses another heuristic, where the idea is again to not need the composition constraint. We found that the idea behind the heuristic is wrong, but we still present it as the idea looks attractive but does not work.

The model checks on the outflow of a charging tank if the transferred volume is within the property specifications. When the composition constraint is added this will mean that what is in the charging

tank has the same values. But when the composition constraint is left out, there is no constraint that makes sure that what is in the tank right before an outflow transfer occurs is within specifications. Thus constraints were added to only allow an outflow out of a charging tank when the crude blend in the tank is within specifications. The reason that this does not work is because this is only correct as long as the outflow does not break the composition constraint. As soon as this happens the newly added constraint will look at crude ratios that are already wrong so even if they are within specifications then that does not mean that what should be in the tank is also within specifications.

We will go through a small example to try to clarify. Figure 13 shows a small network to illustrate the example. The network has two storage tanks, where the first storage tank $S1$ has a crude type A with property value 0.2 and the second storage tank $S2$ has a crude type B with property value 0.3. We will for this example assume that the storage tanks have infinite volume. Then there is a charging tank $C1$ which has an initial volume of 100Mbbl of crude type A and has the property specification $[0.17, 0.24]$.

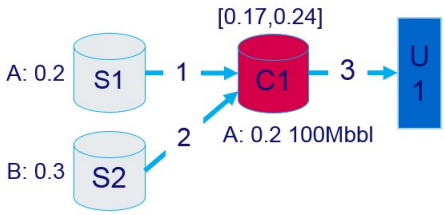

Figure 13: A small network to illustrate the example of why the charging tank specification heuristic does not work

Now we look at Table 4, which shows us a possible outcome of choices that the optimizer made. First there is a transfer of 40 Mbbl of crude type B to the charging tank. Next there is an outflow from the charging tank of 100 Mbbl, where 60 Mbbl is of crude type A and 40 Mbbl is of crude type B. Note that this is against the composition constraint. But the property value in the charging tank was 0.229, which is within the specification and thus allows for the outflow, and the property value of the outflow is 0.24, which is also within the specification. Note that after the outflow there is 40 Mbbl of crude type A left in the charging tank. Next there is a transfer of 20 Mbbl of crude type A and 40 Mbbl of crude type B to the storage tank. This bring the property value within the charging tank to 0.24, which is again within specifications. Finally there is an outflow from the charging of 100 Mbbl which empties the tank, the outflow had of course a property value of 0.24.

If we now take the same sequence of events but this time adhere to the composition constraint we will see what goes wrong. The outflow of the first 100 Mbbl will consist of 71.43 Mbbl of crude type A and

Table 4: Example of possible optimizer choices to show why the charging tank specification heuristic does not work

Operation (From-To)	Volume Transferred (Mbbl)
S2-C1	40
C1-U1	100 (A: 60, B: 40)
S1-C1	20
S2-C1	40
C1-U1	100 (A: 60, B: 40)

28.57 Mbbl, which has the same property value as what was in the tank, namely 0.229. So far there is nothing wrong. Note that there is now 28.57 Mbbl of crude type A and 11.43 Mbbl of crude type B left in the charging tank. Next we get the transfers bringing the volumes to 48.57 Mbbl of crude type A and 51.43 Mbbl of crude type B. The property value in the charging tank now became 0.25, which is outside of the specifications. Note that this is the same property value which will flow out of the charging tank when it empties. Thus this example shows that this proposition will not work.

4.5 LOCALSOLVER

The LocalSolver model we use is the same as the CP model, as it is also used as the second step in the two-step approach. The model used for the first problem instance is given in Appendix 13. In Section 5.3.6 it is shown that this model does not perform well, so we have limited the attention we pay to it here.

4.6 IMPROVING PERFORMANCE

This section will propose several ways to improve the performance of Mouret's MIP and the Constraint Programming model, and will also indicate some ways that could possibly have improved the performance but did not.

4.6.1 *Improving Mouret's MIP Model*

This section proposes various ways to improve Mouret's MIP model and also has a subsection on ways that failed to improve the model.

4.6.1.1 *Decrease Crude Variables*

Less crude volume variables obviously implies a reduced search space. This method is already used by using the assignments made by the

MIP, but the number of crude volume variables can be further reduced. The model as presented gives an operation the possibility to transfer all the possible crude oils, but in reality and enforced by the level constraints this is not the case most of the time. For instance a vessel arriving with one type of crude can obviously only unload that type of crude.

We therefore propose to decrease the crude volume variables by only creating crude volume variables for an operation if the resource the volume is transferred from has the possibility of having that crude in storage. It is possible for a crude to be in storage at a certain resource r, if there is a path of operations from a resource that has the crude as initial crude to r. This is calculated, in a declarative way, for every resource $r \in R$ by defining a set *allowed crude oils* that has the start crude of r together with the crude sets of all resources that have an outlet operation to r, of course the crude oils in the set are kept unique, so there will be no duplicates.

4.6.1.2 *No Derived Variables*

Another improvement is to only use the decision variables, which takes away readability of the model but removes all the derived variables, which could improve performance. This is partly tried in this thesis by replacing the total volume variable by the sum of its respective crude volume variables.

4.6.1.3 *No Consecutive Assignments*

An operation can be assigned to consecutive priority slots, but this does not add possibilities because if that happens then they can be merged into one operation. To remedy this, constraint 4.25 is added, expressing that an operation that may not overlap with itself cannot be assigned to two consecutive priority slots.

$$Z_{iv_1} + Z_{(i+1)v_1} \leq 1 \qquad i \in T, i \neq n, v_1 \in W, OM_{v_1v_1} = 0 \tag{4.25}$$

4.6.1.4 *No Transfer Operations at the End*

Another improvement is adding constraint 4.26, which forbids the assignment of transfer operations on the last priority slot (i. e., n). A transfer operation is only used as preparation to create space for unloading vessels or to have a crude blend in the charging tanks. But there is obviously no unloading or charging operation after the last priority slot. Thus we know that no solution is lost by removing the possibility of assigning transfer operations on the last priority slot.

4.6.1.5 *No Strengthening Constraints*

In Section 4.1.6.2 the strengthening constraints of Mouret were presented. They were introduced as extra constraints to improve perfor-

mance, however, we found these constraints to be more of a detriment to the performance than an improvement. For this reason, to improve performance, we removed these strengthening constraints in our model.

$$Z_{iv} = 0 \qquad i \in T, i = n, v \in W_T \tag{4.26}$$

4.6.1.6 *Failed Improvements*

This section will present some ways that were thought to be improvements but after running the tests the results showed that the performance decreased.

Flowrate Lowerbound

There are constraints that give lower and upper bounds to the flowrate of an operation. But the flowrate can be given a direct value for most operations. The distillation operations have the constraint that they need to continuously charge the CDUs, thus here it is important to be able to set different flowrates. The other operations have no reason to use a flowrate under the maximum. Thus the flowrate bound constraints can be changed such that for operations other than distillation operations are set to the maximum flowrate.

Symmetry Breaking in the Objective Function

Another way to add symmetry breaking, besides the symmetry constraint presented in Section 4.1.6.1, is to change the objective function. This is done by giving the priority slots a score from high to low or from low to high. A score is counted if an operation is assigned. Using this approach the solver will try to assign an operation to the earliest priority slot to get the highest score added or the lowest score subtracted. If high to low score is used then the score is added to the objective value. Note that this change will also try to get as many assignments as possible after the objective of maximizing the total gross margin. If low to high is used the score is subtracted from the objective value. Note that this change will also try to keep assignments to a minimum. It is important to keep the weight of the total gross margin higher as that objective is more important. Objective function 4.27 shows the high to low variant and objective function 4.28 shows the low to high variant.

$$maximize \sum_{i \in T} \left(\sum_{r \in R_D} \sum_{v \in I_r} \sum_{c \in C} (G_c \cdot V_{ivc}) + \sum_{v \in W} (Z_{iv} * 0.00001 * (n - i)) \right) \tag{4.27}$$

$$maximize \sum_{i \in T} \left(\sum_{r \in R_D} \sum_{v \in I_r} \sum_{c \in C} (G_c \cdot V_{ivc}) - \sum_{v \in W} (Z_{iv} * 0.00001 * i) \right) \tag{4.28}$$

4.6.2 *Constraint Programming Performance*

For the CP model we studied three points that can influence performance besides the other topics we just discussed in this section.

- The first point is the size of the domains. The domains influence the precision of the search. The larger the domain, the larger the search space, allowing for possibly new solutions. The scheduling horizon H has a granularity of seconds but that could also be minutes which takes away a factor 60 of possible values from the time domains. On the other hand, this decreases the precision of the search and could possibly have a worse final solution than when using seconds. It is clear that there is a balance between good solutions and performance. The same holds for the quantity domains (volume and level). Instead of using Mbbl it could for instance be set to bbl which increases the domains by a factor 1000, decreasing performance but with a possibility of finding better solutions.
- The second point is the search strategy. The way the search is performed influences the performance significantly, which is why it is important to discover how to best configure the search.
- The third point is that setting an upper and/or lower bound on the objective value could enhance performance for the CP. The CP's upper bound can be set to the objective value found by the MIP, as it is impossible to find a better objective value when the composition constraint is added.

5 COMPUTATIONAL STUDY FOR THE COSP

This chapter presents the computational study we did for the COSP.

Section 5.1 presents the different problem instances that were used to test the implemented models. Section 5.2 compares our implementation of Mouret's MIP with Mouret's own results, after which Section 5.3 shows the different experiments and their results. This starts with Section 5.3.2, which compares Mouret's original model, the "Cardinality Rule" model, and our"Improved" model. Next there is Section 5.3.3, which looks at our "Improved" model as a possible heuristic. After that we take a look at the results of the 1-Step Heuristic in Section 5.3.4. Section 5.3.5 looks at the results of the CP model and Section 5.3.6 presents the results of LocalSolver. We also take a look at the NLP solver named CONOPT that was used in Mouret 2010 [15] as the second step in Section 5.3.7. Lastly a conclusion of the experiments is given in Section 5.4.

5.1 PROBLEM INSTANCES

Four of the problem instances used were taken from Lee et al. 1996 [11]. They are named COSP1 to COSP4. These instances were also used by Mouret [15] making them ideal to be used as benchmarks. To increase the practical relevance of our study we added 3 new instances COSP5-7 which are larger and have different types of vessels. We next added 13 new instances, all being variations of COSP1-7, this to get a broader set of instances upon which we can judge the performance of the different approaches.

The instances are defined by their network, the resources and paths between resources, and input data. Table 5 refers to the respective network figures and data tables. Note that, other than COSP1, the problem instances can be found in Appendix 17.

Table 5: COSP problem instances

Problem instance	Network	Data
COSP1	Figure 14	Table 7
COSP2	Figure 23	Table 21
COSP3	Figure 23	Table 22
COSP4	Figure 24	Table 23
COSP5	Figure 25	Table 24
COSP6	Figure 26	Tables 25 and 26
COSP7	Figure 27	Tables 27 and 28

The 13 variation instances are listed in Table 6. There are three groups of variants. The first group varies the number of properties that are considered, to see what the effect is of having more or less properties. The second group sets all the minimal capacities of the resources to zero, to see if this has any effect on the 1-Step Heuristic. The last group splits the vessels into two equal parts. This means that an original vessel splits into two vessels which are exactly the same (arrival time, operations, etc), except for the volume which is half of the original volume. This is done to see what happens when there are more vessels without changing the instance too much.

Table 6: COSP problem instances variants

Problem instance	Description
COSP2a	Only property 1 is considered
COSP5a	Only property 1 is considered
COSP5b	Only properties 1 and 2 are considered
COSP6a	Only property 1 is considered
COSP6b	Only properties 1 and 2 are considered
COSP7a	Only property 1 is considered
COSP7b	Only properties 1 and 2 are considered
COSP4c	All minimal capacities are set to zero
COSP5c	All minimal capacities are set to zero
COSP6c	All minimal capacities are set to zero
COSP7c	All minimal capacities are set to zero
COSP3v	Vessels are split into two
COSP4v	Vessels are split into two

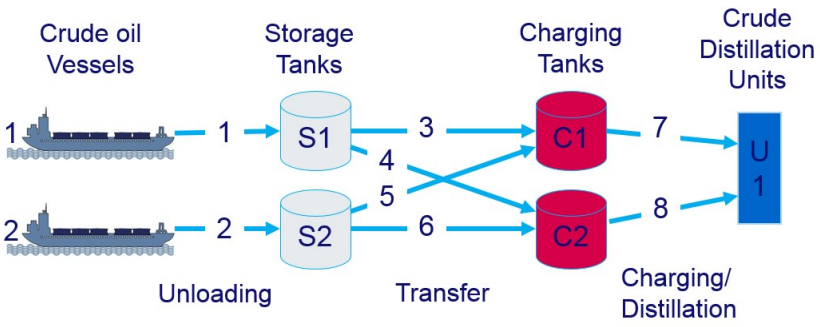

Figure 14: Crude oil operations network for COSP1 from Lee et al. 1996 [11]

Table 7: Overview of the COSP1 data

Scheduling horizon			8 days
Vessels	Arrival time	Composition	Amount of crude (Mbbl)
Vessel 1	0	100% A	1,000
Vessel 2	4	100% B	1,000
Storage tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 1000]	100% A	250
Tank 2	[0, 1000]	100% B	750
Charging tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]	100% C	500
Tank 2 (mix Y)	[0, 1000]	100% D	500
Crudes	Property 1 (sulfur concentration)		Gross margin ($/bbl)
Crude A	0.01		1
Crude B	0.06		6
Crude C	0.02		2
Crude D	0.05		5
Crude mixtures	Property 1 (sulfur concentration)		Demand (Mbbl)
Crude mix X	[0.015, 0.025]		[1000, 1000]
Crude mix Y	[0.045, 0.055]		[1000, 1000]
Unloading flowrate	[0, 500]	Transfer flowrate	[0, 500]
Distillation flowrate	[50, 500]	Number of distillations	3

5.2 VERIFICATION AND VALIDATION

The verification of our implementation of Mouret's MIP was done using his computational results which are presented in Table 8. The n column indicates the number of priority slots used. The MIP column shows the objective values of the MIP solutions, the "no improvement" indication means that there is no better solution found compared to 1 priority slot less. The NLP column shows the objective values of the NLP solutions. The Gap column gives the variation of objective values between the best MIP solution and the NLP solution. Our results where exactly the same as reported by Mouret, thus we concluded the MIP was correctly implemented.

The validation has been done using a Quintiq application, which we debugged, slightly improved and changed to allow for the optimizer to run. The application checks if a given solution violates any constraints and gives a nice visual overview.

5.3 EXPERIMENTS AND RESULTS

This section discusses the setup used for the experiments and their results.

Table 8: Computational results taken from Mouret 2010 [15]

Instance	n	MIP	NLP	Gap
COSP1	1-4	infeasible		
	5	7975	7975	0%
	6	no improvement		
COSP2	1-3	infeasible		
	4	9000		
	5	9617	10117	0%
	6	10117		
	7	no improvement		
COSP3	1-2	infeasible		
	3	8250		
	4	8450	8540	2.3%
	5	8740		
	6	no improvement		
COSP4	1-3	infeasible		
	4	13255	13255	0%
	5	no improvement		

5.3.1 *Setup*

The experiments were run on an *Intel(R)Core(TM) i5 CPU M 450 @ 2.40GHz* CPU. The MIP models were implemented using Quintiq software, which uses *IBM ILOG CPLEX Optimizer version 12.5* as solver. The CP was implemented using *IBM ILOG CPLEX CP Optimizer version 12.6*. The LocalSolver tests were run on *LocalSolver 4.5*.

5.3.2 *Comparing Mouret's MIP to the Improved MIP*

Table 9 presents the results of running Mouret's "Dissertation" MIP, the "Cardinality Rule" MIP, and our "Improved" MIP. The "Dissertation" is the model as presented in Mouret 2010 [15]. This model includes the symmetry and strengthening constraints from Sections 4.1.6.1 and 4.1.6.2. The "Cardinality Rule" model adds the cardinality constraints of Section 4.1.6.3, that were found in the online published models [14]. The cardinality rule presented in Section 4.1.6.3 is used to get the minimum and maximum cardinality values. The "Improved" model changed and added several things in Mouret's MIP model to further improve the performance. Note that this model also uses the cardinality constraints and cardinality rule of Section 4.1.6.3. These changes and addition are mostly explained in Section 4.6.1, and can be shortly summarised as follows:

- Only allowed crude volume/level variables, see Section 4.6.1.1

- Total volume replaced with sum of crude volumes, thus removing the total volume variables, see Section 4.6.1.2
- No two consecutive assignments of the same operation, see Section 4.6.1.3
- No transfer operations allowed on the last priority slot, see Section 4.6.1.4
- The strengthening constraints are removed, see Section 4.6.1.5

The n shows the best number of priority slots found. The "Obj. Val." column shows the obtained objective value, "no sol." indicates that there was no solution found within the time limit, also note that the bold objective values are the best found objective values (if the same value is found we look at the best search time) for their respective instance. The "CPU" column denotes the total accumulated search time in seconds. Note that the objective values for COSP1 to COSP4 are the same as given in Table 8, indicating that the models are correctly implemented. The time limit per iteration was set to 90 seconds. The reason is that there are sometimes up to three feasible iterations until no better solution was found, so that if they run up to the time limit we get 4.5 minutes, which is still within the time limit of 5 minutes.

Note that some runs stopped at the time limit of 1.5 minutes. This does not mean that a better result can be found but that the current result is not confirmed to be optimal. These runs do have a relative optimal gap of less than one percent, indicating that the found solutions are good enough to use in practice.

The first thing to notice when looking at the results is an obvious performance improvement when we look from left to right.

When we compare the 'Dissertation' and 'Cardinality Rule' models we see one of two cases. The first case is that we see an improvement in the found objective values. We only see this when the iteration was stopped, for the 'Dissertation' model, on the time limit, which means that the objective value was not proven to be optimal and thus had a large chance to be able to increase a bit more. Sometimes this leads to an extra iteration which increases the running time, but as we find a better objective value this is an improvement. The second case is that we see an improvement of the running time performance when the same objective value is found, except for COSP1. This improvement lies between 1.3 to 9 times faster search times. From this comparison we can conclude that the cardinality constraints improve performance considerably and should have been included in the dissertation, together with an explanation of where the minimal and maximal cardinality values come from. To remedy this we have proposed the cardinality rule, which gives the same values for COSP1-4 as Mouret used in his models found online [14].

Table 9: Mouret's MIP results, comparing dissertation, model with cardinality rule, and the improved model

	Dissertation			Cardinality Rule			Improved		
Instance	n	Obj. Val.	CPU(s)	n	Obj. Val.	CPU(s)	n	Obj. Val.	CPU(s)
COSP1	5	7975	4	5	7975	5	5	**7975**	1
COSP2	6	10117	185	6	10117	54	6	**10117**	9
COSP3	5	8740	99	5	8740	14	5	**8740**	5
COSP4	4	13255	103	4	13255	18	4	**13255**	3
COSP5	6	15641	271	6	15641	199	6	**15651**	111
COSP6	5	27150	182	6	**27159**	273	5	27156	181
COSP7		no sol.	276		no sol.	280	9	**42096**	275
COSP2a	6	10117	203	6	10117	48	6	**10117**	15
COSP5a	6	15745	249	6	15745	191	7	**15751**	197
COSP5b	6	15745	245	7	**15748**	276	6	15745	107
COSP6a	5	27115	181	6	**27159**	273	5	27156	181
COSP6b	5	27104	181	6	27157	272	5	**27161**	180
COSP7a		no sol.	277	8	42080	188	8	**42134**	182
COSP7b		no sol.	276		no sol.	276	8	**42053**	183
COSP4c	4	13261	126	4	13261	14	4	**13261**	5
COSP5c	7	15639	354	6	15651	204	6	**15651**	120
COSP6c	5	27139	180	7	**27161**	362	6	27160	271
COSP7c		no sol.	278		no sol.	278	9	**41997**	273
COSP3v	7	8740	182	7	8740	110	7	**8740**	19
COSP4v	6	13255	182	6	13255	108	6	**13255**	11

Next we compare the 'Cardinality Rule' and 'Improved' models. First note that the 'Improved' model finds solutions for all instances, where the other versions did not. This is a large improvement. It is observed in the larger and thus practically more relevant COSP7 instances, thus indicating the additional practical value of the improved model. We also have the same two cases where we see an improvement of objective value, mostly small changes though COSP7a increased by 54, or total search time. For the search time we improvements between 1.3 up to almost 10 times faster search times. There are even a few cases where we see both, compare for instance the results of COSP5, COSP6b, and COSP7a. We also see that less iterations are needed in some cases, see COSP6, COSP6a, COSP6b, and COSP6c, which improves the search time by 90 seconds, while still finding comparable values. It is important to notice that there is only one occurrence that had a slower search time, see COSP5a, which is only a 6 seconds difference and the 'Improved' version even found a better solution. There are also no occurrences where a higher n was found. From this comparison we can conclude that the 'Improved' version

has better performance than the 'Cardinality Rule' version, meaning that we have further improved the model that Mouret [15] [14] presented. Note that only the 'Improved' version found a solution for each instance within 5 minutes, with only three occurrences where it took over 4 minutes.

When we look at the results of the property variant instances the first thing to notice is that we cannot draw a straight conclusion about the performance when the instance has more or less properties. However we do observe that if less properties are considered then there is the possibility of a higher objective value, as there are less property specification constraints, which is proven by the presented results.

Next, we look at the results of the no minimal capacity instances. Here we do not immediately see a large difference with their respective main instance results, but we can still make some interesting observations. The instances did get a bit more freedom so a higher objective value is possible as we see with the results of COSP4c and COSP6c. We also see lower objective values when the time limit is reached, see COSP7. We also observe an increase in iterations for COSP6c From this we conclude that the extra freedom makes the instance harder to solve.

Lastly, we look at the results of the split vessel instances. First we notice that the best objective values are still the same as for their respective main instance. When we split the vessels there is in essence only one change to the problem which is that a vessel is now unloaded in two steps, thus allowing for more freedom, which could have resulted in a better objective value but certainly not a worse objective. Second we notice that there are more priority slots needed. The reason is that the instances now have 6 vessels which cannot overlap, thus their respective unloading operation needs to be scheduled on a separate priority slot. Meaning that for 6 vessels we need at least 6 priority slots. There is also an increase in the search time which is probably a result of the increased freedom and the increase in priority slots. Note that these instances were created to see what happens if more vessels were added i. e., not directly to allow for more unloading operations per vessel.

We just saw that the larger instances often have no proof of having found an optimal solution before the time limit of 90 seconds. To see how much of a difference it would mean on the objective values, if we had a higher time limit, we ran the same instances, but now with a time limit of 900 seconds. These results are presented in Appendix 15. Note that in terms of quality (objective value) "Cardinality Rule" and "Improved" are quite comparable, but we still see that the "Improved" model has better search times or objective values most of the time. The results further show us differences of less than 1% between the objective values, where most are even less than 0.1%. From this we can conclude that increasing the time limit does not give significantly

better objective values on our instances, thus allowing us to use the time limit of 90 seconds, which gives us total maximal search times of around 5 minutes.

5.3.3 *Improved MIP Heuristic*

Our Improved MIP is used as the first step of the two step approach. The reason that the MIP of itself is not a good solution method is that the non-linear composition constraint is not taken into account. This means that a obtained solution could have an actual objective value that is lower and/or the blends in the charging tanks are outside the property specifications. To see if this is what happens in practice, we took the obtained solutions from our Improved MIP and checked what the actual values are and how far the solutions are outside their respective property specification. The results are presented in Table 10. The "Obj. Value" column shows the objective value of our Improved MIP. The "Act. Value" is the value of the solution if it were to be used in practice, meaning that the composition constraint is adhered to, this value is calculated by the Quintiq application after inputting the found solution. The "Diff" column is the difference of the objective and actual value in percent. The last column "Within Spec." shows how far outside the property specification the found solution really is. For every charging operation we calculate what the percentage is that the blend properties are within the property specifications of the respective charging tank. These percentages are averaged per charging tank, thus the average of all operations that flow out of a certain charging tank. We then take the sum of this percentage multiplied by its respective demand and finally divide by the total demand.

The results present us something interesting. First we see that the actual value is mostly within 1% difference and at most 6.83%. Second is that all solutions are within 1% of a 100% property specification adherence. Even more noteworthy is that 12 solutions have a 100% property specification adherence, meaning that these solutions can be used without any problems. Note that there is no guarantee that a repeat of the same run will find the exact same solution and thus the same actual value, but there is enough evidence to suggest that another solution will also be very close to or exactly be 100% within property specifications. From these results we conclude that our Improved MIP is a heuristic of itself. Note that the total running time is always faster than the two-step approach as there is obviously no second step. However, if the second step is done with an NLP solver (e. g. CONOPT), this is not an improvement as we found the NLP solver will solve the NLP within a second.

Table 10: Mouret's MIP as Heuristic Results

Instance	Obj. Value	Act. Value	Diff.(%)	Within Spec.(%)
COSP1	7975	7975	0	100
COSP2	10117	9780	3.33	100
COSP3	8740	8143	6.83	100
COSP4	13255	13254	0.01	100
COSP5	15651	15555	0.61	100
COSP6	27156	26806	1.29	99.948
COSP7	42096	42042	0.13	99.78
COSP2a	10117	10117	0	100
COSP5a	15751	15690	0.39	100
COSP5b	15745	15655	0.57	100
COSP6a	27156	27154	0.01	99.684
COSP6b	27161	26918	0.89	99.993
COSP7a	42134	42087	0.11	99.092
COSP7b	42053	41837	0.51	99.998
COSP4c	13261	13261	0	100
COSP5c	15651	15555	0.61	100
COSP6c	27160	26984	0.65	99.993
COSP7c	41997	41981	0.04	99.915
COSP3v	8740	8526	2.45	100
COSP4v	13255	13255	0	100

5.3.3.1 *Tighter Specification Bounds Heuristic*

The results of our Improved MIP where we saw that the solutions are very close to being within the bounds gave the idea of tightening the property specification bounds. The thought behind this idea is that if our Improved MIP gives solutions that are almost within the specification bounds, then, if we tighten the specification bounds for the optimizer, the actual solution (e. g. adhering to the composition constraint) will be within the property specifications. A downside of this approach is that there is a chance that the optimal solution cannot be obtained, but a good solution can still be obtained as long as the bounds are not made too tight. Another potential downside is that there may be no feasible solution left because certain crude blends are no longer possible.

This heuristic was tested using three different levels of tightening the bounds, namely 0.5%, 1%, and 3%. This means that a specification lower bound is increased by x% and a specification upper bound is decreased by x%. The results are presented in Table 11.

The first level of 0.5% gave some good results until COSP6 where no feasible solution was left. The reason for this is likely because the start crude oils in the charging tanks are already at the upper bound.

Table 11: Results of the MIP with tightened bounds

Tightening Bounds: 0.5%				
Instance	Obj. Value	Actual Value	Diff.(%)	Within Spec.(%)
COSP1	7936	7913	0.290	100
COSP2	10075	10075	0.000	100
COSP3	8711	8390	3.685	100
COSP4	13221	13221	0.000	100
COSP5	15567	15562	0.032	100
COSP6	Infeasible	-	-	-
Tightening Bounds: 1%				
Instance	Obj. Value	Actual Value	Diff.(%)	Within Spec.(%)
COSP1	7879	8061	-2.310	97.197
COSP2	10032	10032	0.000	100
COSP3	8681	8450	2.661	99.858
COSP4	13187	13275	-0.667	99.51
Tightening Bounds: 3%				
Instance	Obj. Value	Actual Value	Diff.(%)	Within Spec.(%)
COSP1	7740	7721	0.245	100
COSP2	9854	9854	0.000	100
COSP3	8564	8185	4.426	99.344
COSP4	Infeasible	-	-	-

This means that with the decrease the start crude oils are not allowed to flow into a distillation unit, thus the model cannot adhere to the continuous flow constraint for every distillation unit. An idea could be to check this and see how far the upper bound can be tightened, but this will need to be tested in a future project.

When we look at the 1% and 3% level, we immediately see that they give solutions that are not within specifications. From this we can conclude that tightening the bounds too much does not work. Not only do we still get solutions outside the bound, but the solutions have a high chance of being suboptimal.

This heuristic does not have any guarantee of finding a solution within specifications, but the results of the first level do show some promise. It would need a lot of testing in practice to see if there is any real difference in accuracy between the normal bounds and the tightened bounds. As this, as well as the previous idea, could not be further researched for this thesis it is added to the future work section.

5.3.4 *1-Step Heuristic Results*

Tables 12 and 13 present the results of the 1-Step Heuristic, which was proposed in Section 4.3. Table 12 shows the results where the 1-Step Heuristic is used as a replacement of the two-step approach. Table 13 shows the results when the assignments found by our MIP are fixed for the heuristic. In this case the heuristic only replaces the second step of the two-step approach.

The n indicates the number of priority slots used. The "Obj. Value" column shows the obtained objective value. The "CPU" column denotes the search time in seconds. The column "Within Spec." shows how far outside the property specification our 1-Step Heuristic MIP really is. For every charging operation we calculate what the percentage is that the blend properties are within the property specifications of the respective charging tank. The "Act. Value" is the value of the solution if it were to be used in practice, meaning that the composition constraint is adhered to.

Note that if the 1-Step Heuristic has the same objective value as our Improved MIP, making the MIP Gap equal to zero, and the Improved MIP found its optimal solution, then the solution found by the 1-Step Heuristic is the best found solution, because the Improved MIP gives an upper bound for the problem.

When studying Table 12, the first thing we notice is that there are only two solutions that are just outside of the property specifications and there is one instance, COSP7c, where no solution was found at all. The second thing to notice is that all found objective values are within 2% of the objective values found by the Improved MIP in Table 9, except for COSP3 and COSP3v, which has a difference of 5.6%. Note that the best found solution by Mouret for COSP3 was 8540, which brings the found objective values of the 1-Step Heuristic within 3.4% of the best found solution. Also note that for COSP1, COSP2, COSP4, and their respective variants we find the exact same objective values as the best known objective values. The search times are still smaller or around the 5 minutes, but we can clearly see that the 1-Step Heuristic has more difficulties finding the best solution than the Improved MIP. However, it is still an interesting heuristic as it does not need a non-linear solver to get a complete solution for the COSP.

Next we look at the results obtained by using the 1-Step Heuristic with the set assignments, meaning that we first run the Improved MIP and use the assignments of its found solution as input for the 1-Step Heuristic, see Table 13. Here we get comparable results as when we do not use the set assignments. There are two instances that are just outside the property specifications, but this time all instances found a solution. When we look at the objective values we see that most of them are a bit lower than when we do not use the set assignments. Which is logical as the search space is decreased, creating the pos-

Table 12: 1-Step Heuristic results

Instance	n	Obj. Value	CPU(s)	Within Spec.(%)	Act. Value
COSP1	5	7975	3	100	-
COSP2	6	10117	19	100	-
COSP3	4	8250	7	100	-
COSP4	4	13255	4	100	-
COSP5	7	15594	307	100	-
COSP6	5	27038	180	100	-
COSP7	10	42019	273	99.999	42015
COSP2a	6	10117	19	100	-
COSP5a	6	15529	211	100	-
COSP5b	7	15689	302	100	-
COSP6a	5	27032	180	100	-
COSP6b	6	26983	271	100	-
COSP7a	9	42008	274	99.989	42007
COSP7b	8	41493	182	100	-
COSP4c	4	13261	6	100	-
COSP5c	6	15433	215	100	-
COSP6c	5	27053	180	100	-
COSP7c		no solution	270		
COSP3v	7	8250	157	100	-
COSP4v	6	13255	31	100	-

sibility that feasible solutions are removed. Note however, that the objective values are still within a 3% difference with the objective values of the Improved MIP, except for COSP3 and COSP3v. When we look at the search times we see a large difference. The search times should be compared though to the second step algorithms as we used the assignments that were found by the first step. When done by an NLP solver, such as CONOPT, the second step does this within a second for instances COSP1-7. Which means that the heuristic is as fast or sometimes slower than an NLP solver, this together with the fact that the NLP solver has a high chance of finding a better solution makes this approach less appreciated. Though if no NLP solver is available it becomes a lot more interesting. If we for instance compare it with the results of the Constraint Programming approach, see Section 5.3.5, we see that the 1-Step Heuristic with fixed assignments is significantly faster, with comparable objective values.

5.3.5 *Constraint Programming Results*

Table 14 presents the results of the Constraint Programming model using fixed and partly fixed assignments. Note that the CP model

Table 13: 1-Step Heuristic with set assignments results

Instance	n	Obj. Value	CPU(s)	Within Spec.(%)	Act. Value
COSP1	5	7975	0	100	-
COSP2	6	10117	0	100	-
COSP3	5	8450	0	100	8041
COSP4	4	13254	0	100	-
COSP5	6	15354	0	99.977	-
COSP6	5	26761	1	100	-
COSP7	10	41601	9	100	-
COSP2a	6	10117	0	100	-
COSP5a	6	15479	0	100	-
COSP5b	6	15399	0	100	-
COSP6a	5	26400	1	100	-
COSP6b	6	26952	1	100	-
COSP7a	8	41599	1	100	-
COSP7b	9	41063	9	99.855	-
COSP4c	4	13261	0	100	-
COSP5c	7	15449	1	100	-
COSP6c	5	26484	0	100	-
COSP7c	8	41620	2	100	-
COSP3v	7	8740	0	100	8285
COSP4v	6	13255	0	100	-

was run on both variants on all instances, but the table only shows the instance if one of the variants found a solution within the time limit of 60 seconds. This time limit was chosen as our Improved MIP already takes 4 to 5 minutes for the larger instances. For the CP model the search strategy was to focus on the crude volumes first and on the start of the operations second. The search used for the CP model was *Restart*. These choices were made after doing performance tests. The results and conclusions of these tests can be found in Appendix 16.

We also experimented if giving the objective value found by Mouret's MIP as an upper bound has any influence, which results in two lines per instance where one has no upper bound and the other was given the upper bound. The upper bound is shown in column "Upper bound".

The first thing to notice that the only instances that are missing are COSP6, COSP7, and their variants. From this alone we can conclude that the CP model is not the best alternative to an NLP solver as it cannot find a solution for the larger instances within the time limit. However, when we look at the other instances and the objective values that are found we see that they are all within 5% of the upper bound, except for COSP3 and COSP3v with the upper bound, which makes

Table 14: CP results

		Fixed		Partly Fixed	
Instance	Upper bound	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	∞	7970	4.38	7965	13.49
	7975	7950	21.31	**7975**	40.8
COSP2	∞	10006	48.82	10112	48.25
	10117	**10114**	33.96	10034	55.63
COSP3	∞	infeasible	0	**8507**	44.69
	8740	infeasible	0	8294	51.7
COSP4	∞	13247	32.97	**13249**	59.6
	13255	13246	56.94	13236	59.95
COSP5	∞	15092	58.25	no solution	
	15651	**15275**	58.13	no solution	
COSP2a	∞	10106	52.62	10078	58.42
	10117	**10112**	52.25	10004	58.39
COSP5a	∞	**15501**	55.79	no solution	
	15751	15353	59.3	no solution	
COSP5b	∞	**15421**	58.43	no solution	
	15745	15383	58.5	no solution	
COSP4c	∞	13245	59.1	13241	59.65
	13261	13244	36.9	**13259**	54.24
COSP5c	∞	15167	58.59	**15246**	37.96
	15651	15216	52.63	15124	55.75
COSP3v	∞	8408	19.75	**8466**	30.03
	8740	8423	48.97	8130	58.96
COSP4v	∞	13245	57.59	no solution	
	13255	**13254**	48.49	no solution	

them good solutions. Note also that if we compare the objective values of COSP3 and COSP3v with the best known objective value, 8540, they are still within the 5% limit of the research assignment defined in Section 1.3.

When we look at the difference between the fixed and partly fixed variant we see two interesting things.

The first is that where the fixed variant found infeasible for COSP3, the partly fixed found a very good solution. Note that, because we found infeasible for COSP3, we can conclude that the found assignments by the Improved MIP will not always result in a feasible solution for the CP model.

The second is that the partly fixed variant does not find a solution for several of the instances where the fixed variant does. This is because the partly fixed variant has a larger search space, resulting in a possible longer search time before a feasible solution is found.

When we look at the difference of using an upper bound or not using an upper bound, we observe both objective values that got higher as well as lower. As such we cannot draw any conclusions on the use of an upper bound. Though we can see that, with the upper bound, the fixed variant gets higher objective values on most instances, while the opposite happens for the partly fixed variant.

To conclude, the CP model could be used as the second step in the two-step approach, but it will need some real improvements if it is to be used for the larger instances.

5.3.6 *LocalSolver Results*

As explained we also set out to evaluate LocalSolver [3], but in the end only tested on a few problem instances. The reason is that the results, see Table 15, showed rather quickly that going further with this model contained limited promise. The first test was on COSP1, which gave the best known result in 5 seconds. This let us think that LocalSolver could also solve the other problem instances in comparable time, but as can be seen in the results no solution was found for COSP2 or COSP3 within 1 hour. Note that the models used for the other instances are correct. This was tested by adding constraints which set the volume variables using a solution of the CP. The same solution was then found by LocalSolver within 1 minute.

Table 15: LocalSolver results

Instance	Obj. Value	CPU(s)
COSP1	7975	5
COSP2	-	3600
COSP3	-	3600

5.3.7 *CONOPT Results*

In Mouret 2010 [15], CONOPT [1] is used as the NLP solver for the second step of the two-step method. Mouret wrote that a local optimum was found within a second. We did our own experiments to confirm that CONOPT is indeed so fast and also works well on the larger instances. This was done by recreating the model and two-step approach. As it became too much work to prepare all 20 instances, only the main 7 COSP instances were tested. The results are pre-

sented in Table 16. Here we observe that a solution for the second step is indeed found within a second by CONOPT for all instances. More importantly, the objective values are all equal or higher than the ones found by the 1-Step Heuristic or the CP Model. With this performance the use of an NLP solver such as CONOPT is recommended compared to the other two methods.

Table 16: CONOPT results

Instance	Improved MIP	Obj. Value	CPU(s)
COSP1	7975	7975	0.02
COSP2	10117	10117	0.02
COSP3	8740	8540	0.03
COSP4	13255	13255	0.03
COSP5	15651	15636	0.08
COSP6	27156	27087	0.27
COSP7	42096	42065	0.91

5.4 CONCLUSION

In this part of the thesis several viable solutions were found for the COSP, as well as some approaches that did not work. The most important items and conclusions on these are stated in this section.

We first looked at Mouret's MIP where we showed the importance of the cardinality constraints that were omitted in Mouret 2010 [15]. At the same time we presented an Improved MIP model that used Mouret's MIP as basis and showed that it had better performance than the model of Mouret.

The newly proposed 1-Step Heuristic gave very good results even for the larger instances. This is the case for both the normal variant as for the variant where we fix the assignments. The other proposed heuristics, the Improved MIP itself and the tightened bounds heuristic, gave mixed results. The Improved MIP can be outside of specifications but only by a slight margin. There would be no need for a second step, as long as this small margin can be ignored in practice, thus making it a good heuristic under those circumstances. The tightening of bounds showed good results for the lowest level of 0.5%, but not for the other levels. More research needs to be done to see if this heuristic is viable in practice.

The adapted two-step approach, using a CP model for the second step, worked well up until a point where the problem instances became too large to handle. The CP variant, that only partly sets the assignments, was worse in performance compared to the basic CP, but could be helpful finding better solutions for certain instances.

Finally we looked at two commercial solvers LocalSolver and CONOPT, to see how their performance is compared to the 1-Step Heuristics and CP Model. We observed that LocalSolver could be fast as a second step, but will need some more work as we did not get any solutions for later instances. When we used CONOPT as second step we found that it outperformed the other second step approaches, finding the best objective values in less than a second.

Part II

Product Blending and Distribution

6

PROBLEM STATEMENT PRODUCT BLENDING AND DISTRIBUTION

This chapter gives a general description of the Product Blending and Distribution Scheduling Problem (PBDSP) and an abstract model representing this problem. The problem description and mathematical model are inspired by Méndez et al. 2006b [13], but are extended to involve the distribution of the products to the vessels.

6.1 GENERAL DESCRIPTION

The general process of the PBDSP was explained in Section 1.7.3. This section goes into more detail describing the problem.

The central activities in Product Blending and Distribution Scheduling are *blending*, *loading*, and *unloading*. The blending operations are the operations that have a transfer to or from a blend header. They are used to make a transfer from one or several component tanks to a product tank using a blend header. The loading and unloading operations fall under *pipeline operations*, where loading goes from a product tank to a demand vessel through a main pipeline, and unloading goes from a component vessel to a component tank through a main pipeline. The main pipelines (often shortened to pipelines) are defined as resources, meaning that we can define how many operations may use a resource at a time.

The PBDSP can then be described as the problem of determining when operations execute (this can be several times during the schedule), how long the executions take, and how much volume is transferred during each operation, such that the *total profit* is maximized. The profit is defined by the *product revenue* minus the *total component cost*. The product revenue is calculated by the sum of product volumes that are transferred into a product tank times their respective *price*. The total component cost is calculated by the sum of component volumes that are transferred into a blend header times their respective *cost*. Thus the objective will be to create as much (high valued) product as possible (product revenue), while at the same time using the cheapest components to create them (total component cost). Note that in Section 1.7.3 an example of the blending problem was already given, here we saw why choosing the right blend of components while keeping it cheap was not trivial. Figure 15 shows a possible network created with the resources and operations available.

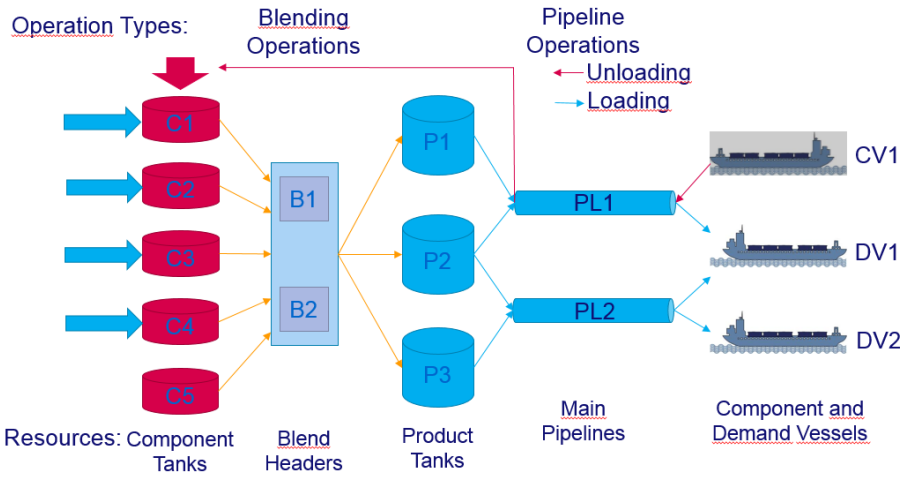

Figure 15: An overview of an example network for the PBDSP

How we define the usage of the pipelines and the rest of the network is stated in the following list of constraints:

- Component tanks can only hold one component.
- Inflow of components is known and constant over time.
- Every component tank is linked to every blend header.
- Every blend header is linked to every product tank.
- Product tanks may only hold one product.
- Product tanks may not have overlapping inflow and outflow operations.
- A pipeline can not be loading and unloading at the same time.
- A pipeline can only have one inflow from a product tank and one outflow to a demand vessel at a time when loading.
- A pipeline can only have one inflow from a component vessel and one outflow to a component tank at a time when unloading.
- A component vessel has one component to unload and the component tank has already been decided.
- A component vessel cannot unload a component in a component tank when that component tank is used for blending.

A PBDSP instance is defined by the following input data:

- A scheduling horizon, indicating the end of the schedule.
- Planned arrival times of component vessels, indicating when a component vessel arrives during the schedule.

- Planned departure times of demand vessels, indicating when a demand vessel departs during the schedule.
- Capacity limits of tanks, indicating how much a tank can hold.
- Transfer flowrate limitations for the blending and pipeline operations, indicating how much crude can, or must in case of a lower bound, be transferred per day.
- Initial composition of tanks and vessels, indicating which component or product is in the tank at the start of the schedule and how much.
- Product specifications for component ratios and properties, indicating that the property values and/or component ratios of the products need to be within the given specifications.
- The cost per component and price per product, which are needed to determine the profit.
- Demand for a product per demand vessel, indicating how much of a product needs to be loaded on a demand vessel.
- Stock of a component per component vessel, indicating how much component volume needs to be unloaded of the component vessel.

6.1.1 *Solution*

Given an instance of the PBDSP, the following decisions need to be made:

- Which product does a blend header produce during a blending operation.
- How much of a component is used by a blend header during a blending operation.
- The time intervals when a pipeline is loading and unloading.
- When and what volume is loaded, from product tanks to demand vessels and through which pipeline.
- When and what volume is unloaded, from component vessels to component tanks and through which pipeline.

Note that this list clearly shows what kind of decisions need to be made but these can also be converted into deciding the following:

- decide how many times a blending, loading, or unloading operation will be executed before the horizon

- decide on the start of each operation execution
- decide on the duration of each operation execution
- decide on the volume transferred during each operation execution.

These decisions are practically the same as for the COSP only we now have three different types of operations, making the problem a bit easier to understand. A solution would again look like Figure 6 only with different operation types.

From these decisions we define a solution to the PBDSP as follows.

We have a set of operations O, which is given by the network, see for example Figure 15 where the arrows represent the operations. We start with defining for each operation $o \in O$ a set of tasks T_o, where the size of T_o, which expresses how often operation o is executed, is defined by $\# : O \to \mathbb{N}$. Let $\mathcal{T}$ be the set of all tasks, i. e., $\mathcal{T} = \bigcup_{o \in O} T_o$. A solution to the PBDSP is then defined as a tuple $< s, d, v >$ where:

- $s : \mathcal{T} \to \mathbb{R}$ gives the start of the task
- $d : \mathcal{T} \to \mathbb{R}$ gives the duration of the task
- $v : \mathcal{T} \to \mathbb{R}$ gives the volume transferred during the task

The problem is then to find a solution that satisfies all constraints of Section 6.1.3 and maximizes the objective function described in Section 6.1.2.

6.1.2 *Objective Function*

The objective of the PBDSP is to maximize the total profit. The total profit is calculated by the product revenue minus the total component cost. The total component cost is the total volume taken from a component tank times its component cost. The product revenue is the total volume of produced product times its price.

Using the solution definition the objective function can be formalized as follows: Let $\mathcal{T}^C$ be the set of tasks of blending operations that transfer volume from a component tank to a blend header and let $\mathcal{T}^P$ be the set of tasks of blending operations that transfer volume from a blend header to a product tank. The total profit can then be obtained by the following equation:

$$totalprofit = \sum_{t \in \mathcal{T}^P} v(t) \cdot price_t - \sum_{t \in \mathcal{T}^C} v(t) \cdot cost_t$$

Where $price_t$ is the price of the produced product of task t and $cost_t$ is the cost of the used component of task t.

We also need to add a penalty to the objective function. Where this penalty precisely comes from will be later explained in Section 7.4,

but is, shortly explained, used for deviation of preferred recipes to give the solution method a bit more freedom. This penalty will be defined as follows:

$$penalty = \sum_{t \in \mathcal{T}^C} excesscost_t \cdot excess_t + shortagecost \cdot shortage_t$$

Where $excess_t$ is the component volume that went over the component volume of the preferred recipe, $shortage_t$ is the component volume that is below the preferred recipe, and $excesscost_t$ and $shortagecost_t$ are their respective cost weights.

The final objective function will then be as follows:

$$objective = totalprofit - penalty$$

6.1.3 *Constraints*

This section gives an overview of the constraints of the PBDSP.

Logistics constraints

A blend header can only produce one product at a time. A product tank can only have an inflow or outflow operation at a time. A pipeline can only load or unload at a time. An unloading operation can only transfer from one component vessel to one component tank. A loading operation can only transfer from one product tank to one demand vessel.

Time constraints

There can be precedence constraints between operations. Boundary constraints can also be added to the operations. For instance a minimum duration constraint to make sure the operation are not to short. Another example is a minimum start that says that the interval of an unloading operation cannot start before the component vessel has arrived.

Tank level constraints

The tank level constraints set the tank levels during the start and end of the operations and check if they are within the capacity bounds of their respective tank. Component tank levels are set by adding the constant inflow times duration plus any unloaded volume minus the volume used for blending. The product tank levels are calculated by adding the produced product volumes minus the volume that has been loaded onto the demand vessels.

Flowrate constraints

The flowrate constraints make sure that the transferred volume can actually flow through the pipelines during the operation's duration. A flowrate constraint can also be added to keep the volume zero when an operation will not be executed.

Product specification constraints

Product specification constraints are added to comply with the product specification. The constraints check if the produced products are within the specified limits. There are two kinds of product specification constraints: *component concentration* and *property specification*. Component specifications are limits on the component ratios within a certain product. If all the component concentrations for a certain product are fixed (i. e., equalizing the upper and lower limit) then it also fixes the recipe of that product, making the property specification constraint redundant as there is no choice left. The property constraints are the limits on the property values of a product. These product property values can be calculated using the component property values and a blending function. The property specification constraints are not straight forward as there are also properties that blend in a non-linear way. See Sections 7.3, 7.5 for more information about properties that blend in a non-linear way.

Unloading and Demand constraints

Unloading constraints are about the unloading of the component vessels, making sure that everything is unloaded. Demand constraints are present if there is a demand given. There are also the constraints that unloading cannot start before arrival of the respective component vessel and loading cannot happen after departure of the respective demand vessel.

Preferred Recipe Deviation constraints

When a preferred recipe is given these constraints allow for deviation but set one of the deviation variables, *excess* or *shortage*, which are used in the objective function. The addition to the objective function will, together with the weight, make sure that the deviations are kept to a minimum.

7

SOLUTION METHODS FOR THE PBDSP

This chapter discusses several important elements of the proposed solution method. Section 7.1 proposes a new time representation that uses Méndez et al. 2006b [13] as inspiration. Section 7.2 explains the recipe optimization. Section 7.3 introduces the linear approximation approach from Méndez et al. 2006b [13]. The iterative procedure proposed by Méndez et al. 2006b [13] is described in Section 7.4. Finally, Section 7.5 introduces blending indices, which are often used in practice.

7.1 TIME REPRESENTATION

In Méndez et al. 2006b [13] a continuous time representation was introduced. It defines subintervals for every unique departure time of the demand vessels, where the subinterval starts at the previous due date and ends at the current due date. For each of these sub-intervals, time slots can be added, which lie within the boundary of the subinterval and do not overlap with each other. These time slots are then used as the scheduling intervals for the blending operations. In every time slot there can be as many blending operations as there are blend headers, but they all use the same time slot interval. Figure 16 gives an overview of this time representation.

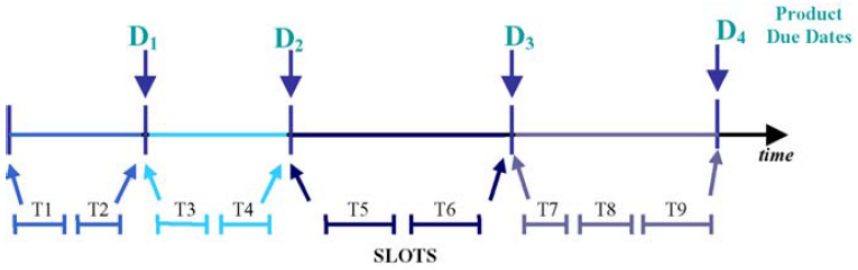

Figure 16: An overview of the continuous time representation by Méndez et al. [13]

This time representation was used by Méndez et al. [13] to solve the *Product Blending Scheduling Problem* (PBSP). This worked as the blending operations were the only operations that were scheduled.

We adapted the time representation to include the pipeline operations, i. e., to solve the PBDSP with the distribution logistics. The adapted time representation has a time slot that encompasses the blending and pipeline intervals. A blending interval represents the time interval for all the scheduled blending operations. The pipeline

intervals are added because of the distribution logistics extension, they represent either a task of an unloading or a loading operation. For every pipeline in the PBDSP an interval is added per time slot. The intervals need to be within their respective time slot, but are independent from each other. Note that the separate intervals do not all have to be used by a schedule, but they are available when needed. Figure 17 shows an overview of the time representation and its usage. The $s(t)$ and $d(t)$ of a task can be derived from the start and duration of the interval. Note that an interval is always used by more than one task, because as was said previously the blend headers and pipelines are not resources that should hold their own volume thus when volume is transferred by a task to one of them then that volume should also be transferred further along to another resource by another task. Thus if there is a task that transfers product volume from a blend header to a product tank during an interval i, then there are also one or more tasks that transfer component volume to the blend header during i.

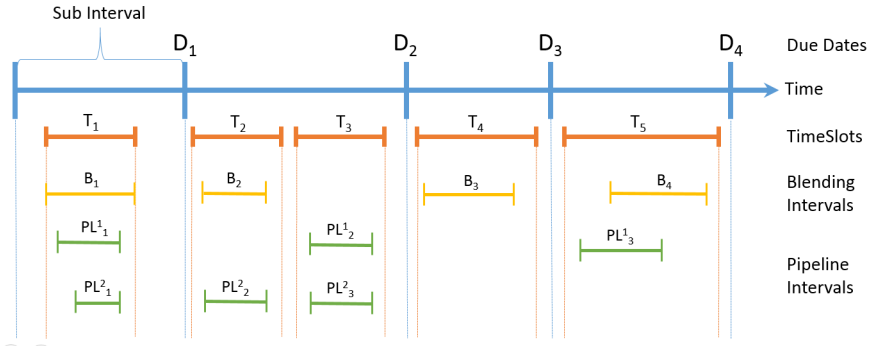

Figure 17: An overview of the continuous time representation

7.2 RECIPE OPTIMIZATION

As mentioned in the introduction, see Section 1.7.3, recipe optimization is about finding the best blending recipe for a product. This is done using only the product specification constraints and a constraint that says that there needs to be 1 bbl of every product produced. The best blending recipe is then the recipe that gives the highest objective value (i. e., most profit). When the optimal product recipes are used as fixed recipes, there is no guarantee that a feasible solution will be found, because there may not be enough components to blend following the optimal recipes. For this reason the optimizer will optimize the recipe and do the logistics scheduling at the same time. This makes the problem harder to solve but will give a better solution. Another approach is to use the optimal recipe as the preferred recipe and allow for deviation. This way the optimizer is encouraged

to use the preferred recipe when possible or use a recipe close to the optimal recipe.

7.3 LINEAR APPROXIMATION

Méndez et al. 2006b [13] presents a solution method to linearize a non-linear blending function in case the correlation for predicting a particular product property is based on a linear volumetric average plus additional non-linear terms. The non-linear terms can then be replaced with a correction factor *bias*. The correction factor is set for every non-linear property for every product for every time slot, using an iterative procedure which will be explained in Section 7.4. The resulting constraints are later presented in Section 8.1.5.

The iterative procedure and the linear approximation proposed by Méndez et al. are illustrated in Figure 18. The figure shows a comparison between the values of the linear volumetric average, the non-linear correlation, and the proposed linear approximation. The example shows the blending of two components A and B, the final product property is a non-linear function of component concentrations. As shown in the figure, if 40% of component A is blended with 60% of component B, the values of the volumetric average and the real non-linear correlation are 88.5 and 88.74, respectively.

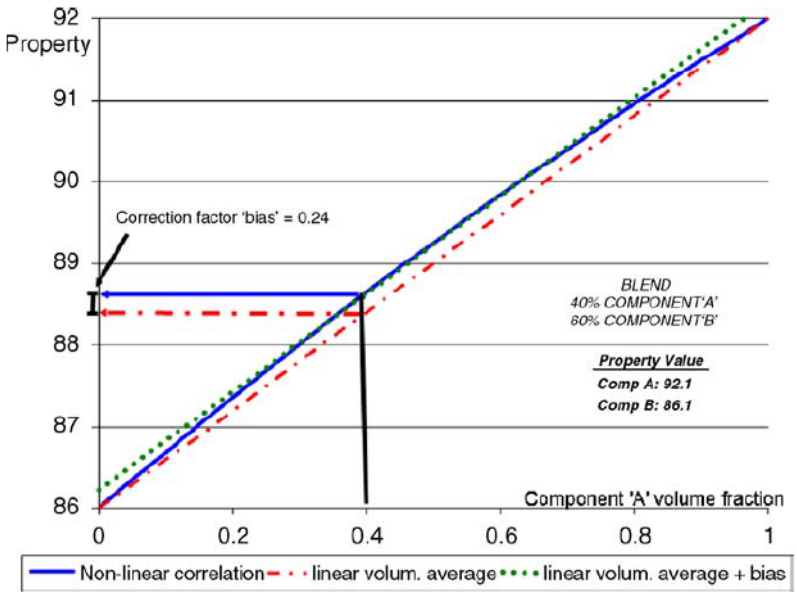

Figure 18: A non-linear property and the proposed linear approximation by Méndez et al. [13]

In order to correct this difference the correction factor *bias* is set to be the difference between the two given values, in this case $bias = 0.24$. The linear approximation comprising the volumetric average together with the correction factor bias will always predict the exact

value of the property if the same component concentration is used in the next iteration. Furthermore, it was observed by Méndez et al. that the proposed linear approximation tends to predict a very close value of the real property if component concentrations are not significantly changed in the next iteration as shown in Figure 18.

7.4 ITERATIVE PROCEDURE

In Méndez et al. 2006b [13] an iterative approach was introduced to solve the PBDSP. This approach is also used for our optimizer as it was proven to be reliable by Méndez et al. Méndez et al. also prove that this approach always converges to an end solution as long as a feasible solution can be found during every iteration. It is however important to note that we changed and extended the model proposed by Méndez et al., which means that this proof might not hold anymore. Because of this there is no guarantee that the iterative approach will converge to a solution, however the results show that the model converges on the given problem instances, see Chapter 9.

Figure 19 shows how the iterative approach works. First an initial recipe is chosen or found for every product, where chosen means that it is manually set by a user, while found means that an algorithm or optimization technique is used, such as recipe optimization of which an example was given in Section 1.7.3. If there are no known preferred recipes, then the simplest method would be to solve the recipe optimization using only the linear properties. This would make the model an easy to solve LP.

The initial recipe is then used to set the initial bias values for every non-linear property for every product for every time slot. The bias is calculated by taking the difference between the non-linear value and the linear value. The non-linear value for a non-linear property for a product is calculated by using the initial recipe and the non-linear blending function. The linear value is calculated using the initial recipe and then taking the volumetric average.

When the initial bias values are set the first run of the optimizer can start. The optimizer can be either an LP model used to find the optimal recipe or a MIP that solves both the recipe optimization and the scheduling problem (i. e., PBDSP). The obtained solution is then used to calculate the actual non-linear values and see if the values are within the specified product limits. If all properties for a certain blending operation are within product specifications than the recipe can be either fixed or set as a preferred recipe, which will still allow for some variation. Note that to solve the PBDSP we will first need to run the LP model to get product recipes after which these recipes are used as initial recipes when we run the MIP model to solve the PBDSP. Both the LP and MIP use the same iterative method to deal with the properties that blend in a non-linear way.

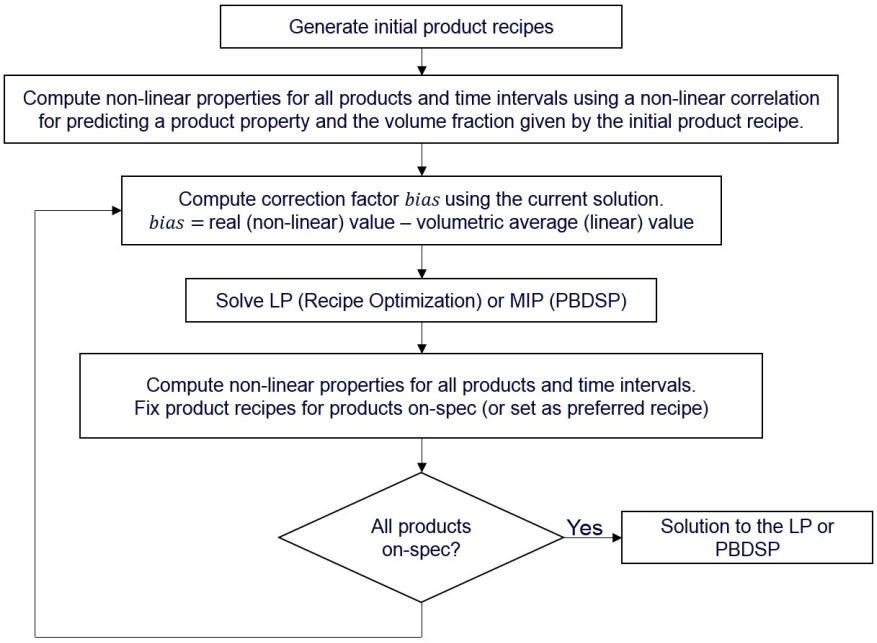

Figure 19: Proposed iterative approach for simultaneous blending and scheduling by Méndez et al. [13]

If all the scheduled blending operations are within product specifications, then a feasible solution is found and the iteration can stop. Otherwise the bias values will be recalculated for all the blending operations that produced product, because if there was nothing produced the bias would always be set to zero. We do not want to reset the bias to zero because, when the bias becomes zero, the non-linear property will be seen as linear and will almost always be outside its specifications, whereas if there is no product produced then the bias will have no influence so it is not important if it is not set to zero. When the bias values are recalculated, the algorithm will execute another iteration.

7.5 BLENDING INDEX

Properties that blend in a non-linear way is what makes the blending problems so difficult. To cope with this problem the refining industry, over a period of many years, has produced many correlations. These correlations have been used to obtain *index functions*, which are used to convert the non-linear properties into *linear indices*. As the name suggests these can be blended in a linear way, making them easy to incorporate in the model as linear constraints. After a solution is calculated, the non-linear property values can be converted back. We will call this the *index approach* in this thesis. We will use this approach as a possible alternative to the iterative approach to deal with the properties that blend in a non-linear way. Figure 20 gives an

overview of how index functions are used. Note that NL is used as an abbreviation of non-linear.

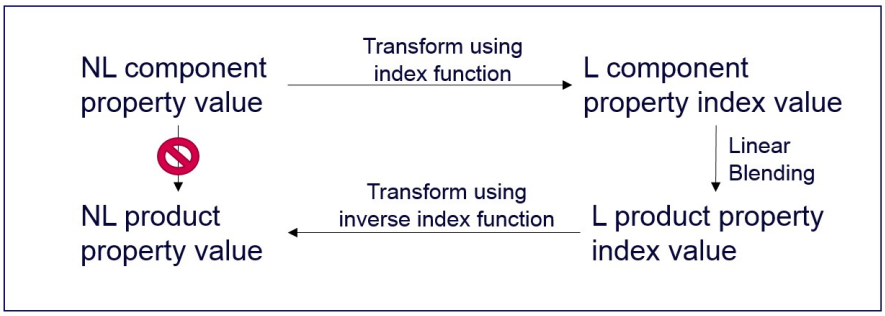

Figure 20: An illustration of the use of index functions

We use the linear indices to see if the property value is within the property specifications. It is therefore important to realise that a maximum specification on some non-linear properties, such as *Flash Point*, has to be converted to a minimum specification on the index when the actual blending calculation is done. However, for other non-linear properties, such as *Reid Vapour Pressure*, a maximum specification on the engineering unit results in a maximum specification on the pour point index.

To further illustrate the use of linear indices two examples are given. The index functions given in the examples are not necessarily true for every refinery.

7.5.1 *Flash Point*

Flash point (FLS) is the temperature at which a product will self-ignite.

The flash index (FLI) is calculated according to the following equation:

$$log(FLI) = 42.1093 - 14.286 * log(FLS + 460)$$

where FLS is defined in degrees F.

The resulting flash index value is then converted back to flash point using the following equation:

$$FLS = 10^{(42.1093 - log(FLI))/14.286} - 460$$

The result is defined in degrees F.

As mentioned above, flash index decreases with increasing flash point. Hence a minimum flash point specification translates to a maximum flash index specification.

7.5.2 *Reid Vapour Pressure*

Reid Vapour pressure Index (RVI) is calculated from the Reid Vapour Pressure (RVP) using the following equation:

$$RVI = RVP^{1.25}$$

where RVP is Reid vapour pressure in PSI

The resulting Reid vapour pressure index value is then converted back to Reid vapour pressure using the following equation:

$$RVP = RVI^{0.8}$$

where the result is in PSI

We will now use the index functions of RVP to give a small example. So first of all we will use the first function on all RVP properties that we have in the input data. So all RVP property values of every component, together with the RVP property bounds of the product property specifications. Note that these bounds would be switched (minimum becomes maximum and other way around) in the case of Flash Point, but will stay the same for RVP. The now newly obtained index values will replace the old RVP property values in the input data. Thus the model will use the index values that blend linearly. When the solution is obtained we can calculate the index values of the product blends used in the solution. These index values can then be transformed back into actual RVP property values using the second function.

8

SOLUTION MODELS FOR THE PBDSP

This chapter presents the mathematical model we developed to solve the PBDSP. The model was inspired by Méndez et al. 2006b [13], and was further adapted to allow for the scheduling of tasks for the loading and unloading of the demand and component vessels respectively.

Section 8.1 presents the proposed model to solve the PBDSP. Section 8.2 presents the recipe optimization model from Méndez et al. 2006b [13] that can be used in the overall approach. Section 8.3 introduces the proposed model that uses indices to keep the model linear without the need of an iterative approach.

8.1 PBDSP MODEL

The *PBDSP model* is the mathematical model used to solve the PBDSP.

8.1.1 *Sets*

The following definitions are used in the PBDSP model:

- C is the set of component tanks
- CV is the set of component vessels
- DV is the set of demand vessels
- D is the set of unique departure times of the demand vessels
- K is the set of properties for the components and products
- P is the set of final product tanks
- PL is the set of pipelines
- T is the set of time slots
- T_d is the set of time slots of the sub-interval, see Section 7.1, ending at due date d
- TT_d is the set of time slots that occur before the sub-interval that ends at due date d

8.1.2 *Input Data*

The model has the following input data:

- $arrival_{cv}$ is the arrival of component vessel cv
- $stock_{cv}$ is the quantity component vessel cv needs to unload
- $component_{cv}$ is the component component vessel cv needs to unload
- $bias_{p,k,t}$ is the correction factor of the value of property k of the product in product tank p in time slot t
- $cost_c$ is the cost of the component inside component tank c
- d_{dv} is the demand due date for demand vessel dv
- $demand_{dv}$ is the demand of demand vessel dv
- f_c is the constant flowrate into component tank c
- h is the time horizon
- ini_c is the initial inventory level of component tank c
- ini_p is the initial inventory level of product tank p
- L_c^{max} is the maximum storage capacity of component tank c
- L_c^{min} is the minimum storage capacity of component tank c
- L_p^{max} is the maximum storage capacity of product tank p
- L_p^{min} is the minimum storage capacity of product tank p
- n_t^B is the maximum number of blend headers that can be working in parallel in time slot t
- n^{PL} is the number of pipelines available for use
- $plty_{c,p}^{R^-}$ is the penalty weight for taking a lower volume concentration of the component in component tank c for the product in product tank p than indicated by the preferred recipe
- $plty_{c,p}^{R^+}$ is the penalty weight for taking a higher volume concentration of the component in component tank c for the product in product tank p than indicated by the preferred recipe
- $price_p$ is the price of the product in product tank p
- $pr_{c,k}$ is the value of property k for the component in component tank c

- $pr_{p,k}^{max}$ is the maximum value of property k for the product in product tank p
- $pr_{p,k}^{min}$ is the minimum value of property k for the product in product tank p
- $rate_{c}^{max}$ is the maximum flowrate from component tank c
- $rate_{c}^{min}$ is the minimum flowrate from component tank c
- $rate_{pl}^{max}$ is the maximum flowrate of pipeline pl
- $rate_{pl}^{min}$ is the minimum flowrate of pipeline pl
- $rcp_{c,p}$ is the preferred volume concentration of the component in component tank c in the product in product tank p according to the preferred product recipe
- $rcp_{c,p}^{max}$ is the maximum concentration of the component in component tank c in the product in product tank p
- $rcp_{c,p}^{min}$ is the minimum concentration of the component in component tank c in the product in product tank p

8.1.3 *Variables*

The following binary and continuous variables are used in the model:

- $A_{p,t}^{B}$ denotes the binary variable denoting that product for product tank p is blended in time slot t
- $A_{p,pl,dv,t}^{PL+}$ denotes the binary variable denoting that product for demand vessel dv is transferred from product tank p through pipeline pl during time slot t
- $A_{cv,pl,t}^{PL-}$ denotes the binary variable denoting that component from component vessel cv is transferred through pipeline pl to the corresponding component tank during time slot t
- S_{t}^{T} denotes the continuous start variable of time slot t
- E_{t}^{T} denotes the continuous end variable of time slot t
- S_{t}^{B} denotes the continuous start variable of blending in time slot t
- E_{t}^{B} denotes the continuous end variable of blending in time slot t
- $S_{pl,t}^{PL}$ denotes the continuous start variable of pipeline operations for pipeline pl during time slot t

- $E^{PL}_{pl,t}$ denotes the continuous end variable of pipeline operations for pipeline pl during time slot t
- $L^{C}_{c,t}$ denotes the continuous tank level variable of component tank c at the start of time slot t
- $L'^{C}_{c,t}$ denotes the continuous tank level variable of component tank c at the end of time slot t
- $L^{P}_{p,t}$ denotes the continuous tank level variable of product tank p at the end of time slot t
- $L'^{P}_{p,d}$ denotes the continuous tank level variable of product tank p at due date d
- $V^{C}_{c,p,t}$ denotes the continuous component volume variable, which is the volume that is taken from component tank c to product tank p during time slot t
- $V^{P}_{p,t}$ denotes the continuous total product volume variable denoting the volume that is transferred to product tank p during time slot t
- $V^{PL^{+}}_{p,pl,dv,t}$ denotes the continuous volume variable, which is the volume that is transferred from product tank p to demand vessel dv through pipeline pl during time slot t
- $V^{PL^{-}}_{cv,pl,t}$ denotes the continuous volume variable denoting the volume that is transferred from component vessel cv to the corresponding component tank through pipeline pl during time slot t
- $D^{R^{-}}_{c,p,t}$ denotes the continuous component shortage variable denoting that for the product in product tank p there was a shortage of the component in component tank c during time slot t according to the preferred recipe
- $D^{R^{+}}_{c,p,t}$ denotes the continuous component excess variable denoting that for the product in product tank p there was an excess of the component in component tank c during time slot t according to the preferred recipe

8.1.4 *Objective Function*

The objective is to maximize the profit, which means revenue minus the costs. The revenue is calculated by the produced product volumes and their price, where the cost is calculated by the used component volumes and their cost. The penalty can be used when a preferred

recipe has been defined, the solver will then try to use the recipe whenever it is possible.

$$maximize \sum_{t \in T} \sum_{p \in P} \left(price_p \cdot V^P_{p,t} - \sum_{c \in C} cost_c \cdot V^C_{c,p,t} \right) - penalty$$
$$penalty = \sum_{t \in T} \sum_{p \in P} \sum_{c \in C} \left(plty^{R^+}_{c,p} \cdot D^{R^+}_{c,p,t} + plty^{R^-}_{c,p} \cdot D^{R^-}_{c,p,t} \right)$$

8.1.5 *Constraints*

This section contains all the constraints of the model.

Blending constraints

Constraint 8.1 expresses that there will not be more blend assignments per time slot than blend headers.

$$\sum_{p \in P} A^B_{p,t} \leq n^B_t \qquad t \in T \tag{8.1}$$

Product volume constraints

Constraint 8.2 defines that the produced product volume is equal to the sum of used component volumes for every blending operation.

$$\sum_{c \in C} V^C_{c,p,t} = V^P_{p,t} \qquad t \in T, p \in P \tag{8.2}$$

Recipe constraints

Constraint 8.3 ensures that the concentration per component per product is within the given limits.

$$rcp^{min}_{c,p} \cdot V^P_{p,t} \leq V^C_{c,p,t} \leq rcp^{max}_{c,p} \cdot V^P_{p,t} \qquad t \in T, p \in P, c \in C \tag{8.3}$$

Constraint 8.4 expresses that the properties of a blended product are within the property specifications for that product.

$$\begin{aligned} pr^{min}_{p,k} \cdot V^P_{p,t} &\leq \sum_{c \in C} pr_{c,k} \cdot V^C_{c,p,t} + bias_{k,p,t} \cdot V^C_{c,p,t} \\ &\leq pr^{max}_{p,k} \cdot V^P_{p,t} \qquad t \in T, p \in P, k \in K \end{aligned} \tag{8.4}$$

Constraint 8.5a sets the recipe shortage variable by taking the difference between the component amount that is taken and the amount that is preferred. Constraint 8.5b works in the same way but looks at the excess instead of the shortage.

$$rcp_{c,p} \cdot V^P_{p,t} + D^{R^-}_{c,p,t} \geq V^C_{c,p,t} \qquad t \in T, c \in C, p \in P \tag{8.5a}$$
$$rcp_{c,p} \cdot V^P_{p,t} - D^{R^+}_{c,p,t} \leq V^C_{c,p,t} \qquad t \in T, c \in C, p \in P \tag{8.5b}$$

Flowrate constraints

Constraint 8.6a ensures that the flowrate lies within the given flowrate bounds. Note that the minimum flowrate will be ignored when there is no flow as there will not be a feasible solution otherwise. Constraint 8.6b expresses that there is no volume transferred when the blend operation is not assigned. It is important to know that these flowrate constraint are set on the inflow of the blend header. The constraints can also be set on the outflow this will reduce the number of constraints but can only be done if all the inlets of the blend header have suitable flowrate limits.

$$\begin{aligned} &rate_c^{min}(E_t^B - S_t^B) - rate_c^{min} h(1 - A_{p,t}^B) \leq V_{c,p,t}^C \\ &\quad \leq rate_c^{max}(E_t^B - S_t^B) \qquad t \in T, p \in P, c \in C \end{aligned} \tag{8.6a}$$

$$V_{c,p,t}^C \leq rate_c^{max} h A_{p,t}^B \qquad t \in T, p \in P, c \in C \tag{8.6b}$$

Level constraints

Constraint 8.7a sets the component tank level at the end of the time slot. This is done by taking the initial level, adding the produced component volume, and subtracting the used component volumes for blending operations. Constraint 8.7b also sets the component tank level but at the start of the time slot.

$$L_{c,t}^C = ini_c + f_c E_t^T - \sum_{\substack{t' \in T \\ t' \leq t}} \left(\sum_{p \in P} V_{c,p,t'}^C - \sum_{cv \in CV} \sum_{pl \in PL} V_{cv,pl,t'}^{PL^-} \right) \qquad t \in T, c \in C \tag{8.7a}$$

$$L_{c,t}^{\prime C} = ini_c + f_c S_t^T - \sum_{\substack{t' \in T \\ t' < t}} \left(\sum_{p \in P} V_{c,p,t'}^C - \sum_{cv \in CV} \sum_{pl \in PL} V_{cv,pl,t'}^{PL^-} \right) \qquad t \in T, c \in C \tag{8.7b}$$

Constraint 8.8 sets the product tank level at the end of every time slot. This is done by taking the initial level, adding the produced product volumes, and subtracting the volumes that have been loaded onto the demand vessels.

$$L_{p,t}^P = ini_p + \sum_{\substack{t' \in T \\ t' \leq t}} \left(V_{p,t'}^P - \sum_{pl \in PL} \sum_{dv \in DV} V_{p,pl,dv,t'}^{PL^+} \right) \qquad t \in T, p \in P \tag{8.8}$$

Capacity constraints

Constraints 8.9a and 8.9b keep the component tank level variables within the capacity limits of the component tanks.

$$L_c^{min} \leq L_{c,t}^C \leq L_c^{max} \qquad t \in T, c \in C \tag{8.9a}$$

$$L_c^{min} \leq L_{c,t}^{\prime C} \leq L_c^{max} \qquad t \in T, c \in C \tag{8.9b}$$

Constraint 8.10 keeps the product tank level within its capacity bound.

$$L_p^{min} \leq L_{p,t}^{P} \leq L_p^{max} \qquad t \in T, p \in P \tag{8.10}$$

Time and sequence constraints

Constraints 8.11a and 8.11b set the duration of blending and pipeline intervals to zero if the intervals are not used.

$$E_t^B - S_t^B \leq h \cdot \sum_{p \in P} A_{p,t}^B \qquad t \in T \tag{8.11a}$$

$$E_t^{PL} - S_t^{PL} \leq h \cdot \left(\sum_{\substack{dv \in DV \\ p \in P}} A_{p,pl,dv,t}^{PL^+} + \sum_{cv \in CV} A_{cv,pl,t}^{PL^-} \right) \qquad t \in T, pl \in PL \tag{8.11b}$$

Constraint 8.12 sets the time sequence between the time slots, making sure that the next time slot does not start until the previous one has ended.

$$E_t^T \leq S_{t+1}^T \qquad t \in T \tag{8.12}$$

Constraints 8.13a and 8.13b express that time slots of a certain sub interval lie within that sub interval, thus the time slots start after the previous sub intervals due date and end before the due date of their own sub interval.

$$S_t^T \geq d - 1 \qquad t \in T_d \tag{8.13a}$$

$$E_t^T \leq d \qquad t \in T_d \tag{8.13b}$$

Constraints 8.14a, 8.14b, 8.14c, and 8.14d keep the blending and pipeline operation intervals within their time slot interval.

$$S_t^T \leq S_t^B \qquad t \in T \tag{8.14a}$$

$$E_t^T \geq E_t^B \qquad t \in T \tag{8.14b}$$

$$S_t^T \leq S_{pl,t}^{PL} \qquad t \in T, pl \in PL \tag{8.14c}$$

$$E_t^T \geq E_{pl,t}^{PL} \qquad t \in T, pl \in PL \tag{8.14d}$$

The following constraints are constraints on the pipeline assignments.

Pipeline constraints

Constraint 8.15a expresses that there cannot be more pipeline loading assignments then there are pipelines during a time slot. Constraint 8.15b says that there can only be one pipeline loading assignment per demand vessel per time slot. Constraint 8.15c says that

there can only be one assignment (blending or loading) per product tank per time slot. Constraint 8.15d says that there can only be one assignment (loading or unloading) per pipeline per time slot. Constraint 8.15e expresses that there can only be one unloading assignment per component vessel.

$$\sum_{p\in P}\sum_{pl\in PL}\sum_{dv\in DV} A^{PL^+}_{p,pl,dv,t} \leq n^{PL} \qquad t \in T \tag{8.15a}$$

$$\sum_{p\in P}\sum_{pl\in PL} A^{PL^+}_{p,pl,dv,t} \leq 1 \qquad t \in T, dv \in DV \tag{8.15b}$$

$$A^{B}_{p,t} + \sum_{pl\in PL}\sum_{dv\in DV} A^{PL^+}_{p,pl,dv,t} \leq 1 \qquad t \in T, p \in P \tag{8.15c}$$

$$\sum_{cv\in CV} A^{PL^-}_{cv,pl,t} + \sum_{p\in P}\sum_{dv\in DV} A^{PL^+}_{p,pl,dv,t} \leq 1 \qquad t \in T, pl \in PL \tag{8.15d}$$

$$\sum_{pl\in PL} A^{PL^-}_{cv,pl,t} \leq 1 \qquad t \in T, cv \in CV \tag{8.15e}$$

Flowrate constraints

Constraints 8.16a and 8.16b are again flowrate constraints but this time for the pipeline loading operations instead of the blending operations.

$$\begin{aligned} &rate^{min}_{pl}(E^{PL}_{pl,t} - S^{PL}_{pl,t}) - rate^{min}_{pl} h(1 - A^{PL^+}_{p,pl,dv,t}) \leq V^{PL^+}_{p,pl,dv,t} \\ &\quad \leq rate^{max}_{pl}(E^{PL}_{pl,t} - S^{PL}_{pl,t}) \qquad t \in T, dv \in DV, pl \in PL, p \in P \end{aligned} \tag{8.16a}$$

$$V^{PL^+}_{p,pl,dv,t} \leq rate^{max}_{pl} h A^{PL^+}_{p,pl,dv,t} \qquad t \in T, dv \in DV, pl \in PL, p \in P \tag{8.16b}$$

Constraints 8.17a and 8.17b are again flowrate constraints but now for the pipeline unloading operations.

$$\begin{aligned} &rate^{min}_{pl}(E^{PL}_{pl,t} - S^{PL}_{pl,t}) - rate^{min}_{pl} h(1 - A^{PL^-}_{cv,pl,t}) \leq V^{PL^-}_{cv,pl,t} \\ &\quad \leq rate^{max}_{pl}(E^{PL}_{pl,t} - S^{PL}_{pl,t}) \qquad t \in T, cv \in CV, pl \in PL \end{aligned} \tag{8.17a}$$

$$V^{PL^-}_{cv,pl,t} \leq rate^{max}_{pl} h A^{PL^-}_{cv,pl,t} \qquad t \in T, cv \in CV, pl \in PL \tag{8.17b}$$

Vessel constraints

Constraint 8.18 ensures that a component vessel has unloaded all the bought component quantity within the horizon.

$$\sum_{t\in T}\sum_{pl\in PL} V^{PL^-}_{cv,pl,t} = stock_{cv} \qquad cv \in CV \tag{8.18}$$

Constraint 8.19 ensures that there is no component volume unload from a component vessel before it has arrived.

$$S^{PL}_{pl,t} + h\left(1 - A^{PL^-}_{cv,pl,t}\right) \geq arrival_{cv} \qquad t \in T, cv \in CV, pl \in PL \tag{8.19}$$

Constraint 8.20 ensures that the demand for a demand vessel is loaded before the due date.

$$\sum_{t \in TT_{d_{dv}}} \sum_{p \in P} \sum_{pl \in PL} V^{PL^+}_{p,pl,dv,t} = demand_{dv} \qquad dv \in DV \tag{8.20}$$

Constraint 8.21 ensures that there is no new product volume loaded onto a demand vessel after the due date, as by then the vessel has already left.

$$\sum_{\substack{t \in T \\ t \notin TT_{d_{dv}}}} \sum_{p \in P} \sum_{pl \in PL} V^{PL^+}_{p,pl,dv,t} = 0 \qquad dv \in DV \tag{8.21}$$

8.2 RECIPE OPTIMIZATION MODEL

As explained in Section 7.2, the *Recipe Optimization Model* only looks at the product specifications to find the cheapest recipes for the products. This means that we only need the constraints 8.3 and 8.4 of the PBDSP model. Furthermore, we add a constraint that limits the production per product to one volume and only use one time slot. The added constraint looks as follows:

$$V^P_{p,t} = 1 \qquad p \in P, t \in T$$

Note that we do not have initial recipes, to solve this problem we first do a run where the non-linear properties are ignored. The recipes obtained using this run are then used as initial recipes for the rest of the iteration where we do look at the non-linear properties.

The recipes obtained by this model can then be used as initial recipes when using the iterative approach with the PBDSP Model.

8.3 BLENDING INDEX MODEL

The *Blending Index Model* is a variant of the PBDSP Model. This model does not need the iterative approach to solve the problem of properties that blend in a non-linear way. It uses preprocessing of the property and product specification values to obtain index values that blend linearly. This method was explained in Section 7.5. To obtain the Blending Index Model we only need to replace constraint 8.4 with the following constraint:

$$\begin{aligned} indexpr^{min}_{p,k} \cdot V^P_{p,t} &\leq \sum_{c \in C} indexpr_{c,k} \cdot V^C_{c,p,t} \\ &\leq indexpr^{max}_{p,k} \cdot V^P_{p,t} \qquad t \in T, p \in P, k \in K \end{aligned}$$

Where $indexpr$ is the index value of pr. Note that, as explained earlier, $pr^{min}_{p,k}$ and $pr^{max}_{p,k}$ could become $indexpr^{max}_{p,k}$ and $indexpr^{min}_{p,k}$ respectively. As the index function may switch the lower and upper bound, look for instance back to the Flash Point property in Section 7.5.1.

9 COMPUTATIONAL STUDY FOR THE PBDSP

This chapter presents the problem instances used for the experiments, the experiments that were used to validate the proposed models, and the results of the experiments. Section 9.1 presents the different problem instances that were used to test the implemented models. Section 9.2 shows the different experiments and their results. Note that the computational study is not an in depth study as done for the COSP. The reason is that the PBDSP was a more direct practical problem that had to be solved and because of time constraints it was decided to only do a more in depth study for the COSP.

9.1 PROBLEM INSTANCES

There are two problem instances used for this computational study. The first is an instance taken from Quintiq's practice, called PBDSP1. The second instance is an extended and adapted version of the problem instance presented in Méndez et al. 2006b [13], called PBDSP2. There are only two problem instances as the primary research focus lies in the COSP problem, whereas for the PBDSP the focus lay in solving PBDSP1. This means that we the conclusions we make will not have a lot of backing, but they can be used to create new theories, which can be further researched in future work. We will also see that the two instances are very different from each other. PBDSP1 has more operation restrictions and only a few properties of which only one is non-linear. The reason for this is that we only have a part of the property data, the rest is held confidential by a potential customer of Quintiq. PBDSP2 has less operation restrictions but a lot more properties of which 4 are non-linear.

The instances are defined by their network, which encompasses the resources and paths between resources, and input data, which has all the needed input values. The data is split into several tables. Table 17 refers to the respective network figures and data tables. The non-linear properties in table "Component Data" are indicated by (NL) behind the property name.

Table 17: PBDSP problem instances

	Problem Instance	
	PBDSP1	PBDSP2
Network	Figure 21	Figure 22
Component Tank Data	Table 35	Table 41
Component Data	Table 36	Table 42
Product Tank Data	Table 37	Table 43
Product Data	Table 38	Table 44
Main Pipeline Data	Table 39	Table 45
Demand Vessel Data	Table 40	Table 46
Component Vessel Data	-	Table 47

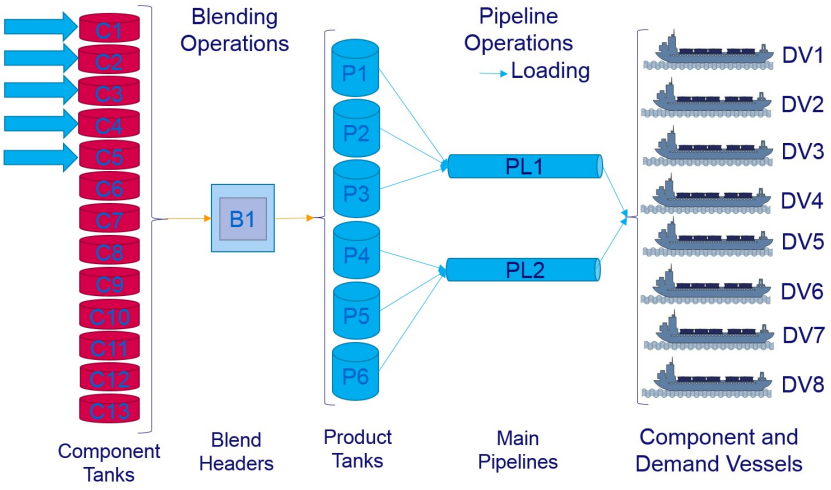

Figure 21: PBDSP1: network

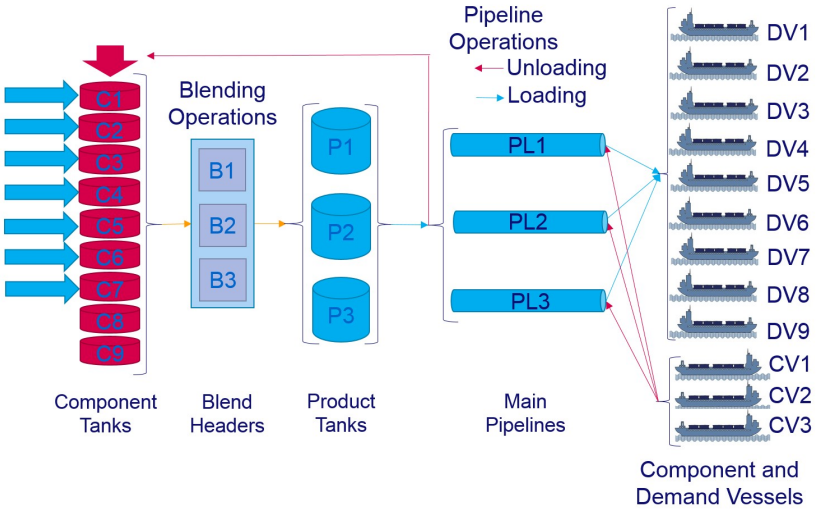

Figure 22: PBDSP2: network

9.2 EXPERIMENTS AND RESULTS

This section discusses the setup used for the experiments and their results.

9.2.1 *Setup*

The experiments where run on an *Intel(R)Core(TM) i5 CPU M 450 @ 2.40GHz* CPU. The models were implemented using Quintiq software, which uses *IBM ILOG CPLEX Optimizer version 12.5* as solver.

9.2.2 *Recipe Optimization Results*

This section presents the results of the Recipe Optimization. Two variants were used, one variant uses the iterative approach and the other uses the index approach. The results are presented in Table 18. The objective value is shown in column "Obj. Value", the CPU time is shown in seconds in column "CPU", and the column "Opt. Gap" gives the relative optimality gap for the index approach.

Table 18: Recipe Optimization Results

	Iterative			Index		
	Obj. Value	CPU	Iterations	Obj. value	CPU	Opt. Gap (%)
PBDSP1	66111.11	0.08s	0	66111.11	0.06s	0
PBDSP2	13393.13	0.6s	7	13393.27	0.05s	0

The Recipe Optimization model is an LP, which can easily be solved as seen in the results, as it finds the optimal solution within a second for both problems. The results further show that the iteration and index approach both give almost the same objective value. Both approaches also give an optimality gap of 0 but as the iteration approach does not give a guarantee on optimality we only show the gap obtained with the index approach. That the iterative approach does not guarantee optimality is directly proven by the results where we see that it found a slight lower objective value. This happens because we can only approximate the actual non-linear blending function using the bias. For the PBDSP1 the initial recipe run, which only looks at linear properties, gave recipes that are also within specifications for the non-linear property thus no iterations were needed.

9.2.3 *PBDSP Results*

This section looks at the results of the PBDSP Model and the Blending Index Model, which use the iterative approach and index approach, respectively. The results are presented in Table 19.

Table 19: PBDSP Results

	Iterative			Index		
	Obj. Value	CPU	Iterations	Obj. Value	CPU	Opt. Gap (%)
PBDSP1	2372.31	536s	33	2372.99	5.88s	0
PBDSP2	2633.60	1.79s	1	2633.60	2.36s	0

When looking at the objective values we again see little difference between the two approaches, but the solving time shows something interesting. For BPDSP1 we see that the blending index approach is almost 100 times faster than the iterative approach, whereas for PBDSP2 we see that the iterative approach is slightly faster. A potential explanation lies in the difference of the properties. As shown in the problem instances, PBDSP1 only has four properties of which one, $P4$, is non-linear, and if looked closely we see that $P2$ does not have any influence at all. Furthermore, tests have shown that $P4$ has a lot of influence on both the recipes and the objective value. This means that it takes a long time to get all the correct bias values, which also explains the large number of iterations. There is also the difference of horizon and time slots. PBDSP1 needs at least 20 time slots, where PBDSP2 only needs 13 time slots, and tests have indicated that more time slots increases the solving time. Lastly is the penalty weight used for the preferred recipe. Here tests have indicated that the weight can have influence on the search time, though not on the profit value.

Besides the huge solving time difference we do see that the index approach solves the problems in a few seconds and always has the highest objective value between the two approaches. This indicates that the index approach is better than the iterative approach, though the negative side is that we need an index function for every non-linear property, which could be a problem in practice.

Lastly it is important to note that although the optimality gap is 0, this does not mean that an optimal solution is found. The reason is because of the time slots. Although small tests have indicated that no better solution is found when using more time slots, see Table 20, there is no guarantee that there is no better solution.

Table 20: Results using large amount of time slots

	Index			
	Obj. Value	CPU	Time Slots	Opt. Gap (%)
PBDSP1	2372.99	157s	102	0
PBDSP2	2633.60	77s	98	0

9.3 CONCLUSION

We cannot draw strong conclusions from the computational study. The reason is because we only looked at two instances which do not give a lot of data. We did not create more instances as creating these instances is not trivial, as shown by the large amount of data needed for the two instances, together with the limited time, and focus on the COSP. However, because of the differences between the problem instances and results, we were able to have some interesting observations, which were enough to create some theories. These theories can be further researched in a later project, see Section 10.2 about future work. It is also important to note that, although the scientific value is low, the value for Quintiq was quite high as they now have a working model with which to test and show in practice.

The conclusion we can draw is that most of the results obtained were within the given limits, except the solving time of almost 9 min. Furthermore, we saw that even when using around a 100 time slots the objective values practically stayed the same, which shows that adding a lot of time slots does not necessarily help find a better solution.

Another conclusion we can draw is that the blending index approach has better performance than the iterative approach, however it has the disadvantage that we need to have the corresponding blending index functions.

Part III

Postlude

10 CONCLUSIONS AND FUTURE WORK

This chapter gives the final conclusions in Section 10.1 and presents some ideas for future research in Section 10.2.

10.1 CONCLUSIONS

In this thesis we studied two refinery scheduling problems, namely the COSP and the PBDSP. We present the conclusions on these two problems in Section 10.1.1 and Section 10.1.2 respectively.

10.1.1 *COSP*

For the COSP we studied several solution methods. We took the two-step approach and model from Mouret 2010 [15] as a starting point. Next we improved the model in various ways as well as created a cardinality rule to get the needed input data. We also looked at the model as a heuristic itself where we saw that the found solutions are already (almost) usable. From here we looked into a 1-Step Heuristic which avoided the need of an NLP solver by avoiding the non-linear constraints. This heuristic gave very good results though not the best. We also looked into the second step, for which we used CP, LocalSolver, and CONOPT. Here we found that CP gave good results for the smaller instances, but broke down on the largest two. LocalSolver only found a solution for COSP1 after which it broke down. Lastly we used CONOPT which by far gave the best results.

To conclude, we found several good solution methods that Quintiq could use to solve the COSP.

10.1.2 *PBDSP*

To solve the PBDSP we studied the solution method of Méndez et al.. They presented an interesting way of dealing with properties that blend in a non-linear way, using an iterative approach. We took this model and adapted it into a new model that has more emphasis on the distribution side, which now includes the loading of demanded final products and the unloading of bought components. We also developed another possible way of dealing with properties that blend in a non-linear way. This way is to use index functions to transform the non-linear properties into linear properties and use these linear properties in the model. There was only a small computational study, as the research emphasis of this thesis lay in solving the COSP. The re-

sults showed that the iterative approach is in some cases slower than the index approach, but both approaches could find good solutions. And even though there is no proof that the iterative approach will converge to a feasible solution, the runs so far all converged.

To conclude, a new problem definition coming from practice was proposed and two approaches to solving the problem with properties that blend in a non-linear way were developed. Both approaches find good solutions, but the index approach has the best performance.

10.2 FUTURE WORK

This section gives ideas on possible future research for both the COSP and the PBDSP.

10.2.1 *COSP*

In this section a number of ideas for future research for the COSP are listed.

- Testing the different approaches on more real-life instances. This will give new insights and more reliable conclusions.
- The Tightening Bounds Heuristic needs more research towards the sensitivity of the levels to see if this approach has any real value. This should include checking how close the properties of the start crude oils are already to the upper and lower bounds of the property specifications.
- For the 1-Step Heuristic one could look into adding cases that allow for partial output when a crude blend is in the tank. For instance, this could possibly be extended by allowing a partial output when there is a certain property value in the tank, which can be forced to be the same property value that flows out of the tank.
- The CP model we created was an adaption of Mouret's model. A potential better approach is to create a CP based scheduling model, where we reason directly with activities, resources, temporal constraints, etc.
- Have a look at Section 3.2.2 which talks about a relaxation method. There was insufficient time to study this for this thesis, but it could be a valid solution method for the COSP.
- One could look at a cost objective function. Costs include for instance: *Sea Waiting Costs*, *Storage Costs*, and *Switching Costs*.
- One could always try to improve the performance of the models, because the refinery business changes constantly which means

that schedules will need to adapt constantly. The faster this can be done, the less pressure there is on the refinery schedulers and they will be more flexible as they can respond faster.

10.2.2 *PBDSP*

In this section a number of ideas for future research for the PBDSP are listed.

- One could start by studying the newly proposed model and see if it can be improved. This could be by adding constraints such as symmetry breaking constraints or an overhaul of the time representation.
- What's more, the model uses time slots which are for now set by the user. It would be better if the time slots are set automatically by an algorithm.
- A next idea is to find or construct more instances to be able to do a full computational study of the different approaches.
- Do a sensitivity analysis on the weight of the preferred recipe, to see how the performance changes.
- The optimal recipe, which is now used as initial recipe, can often not be used in the actual schedule. Thus it would be a good idea to change the optimal recipe model to find a good recipe that has a high chance of being used in the full PBDSP model.
- Study the iterative approach and its convergence. Can it be proven that it always converges? Is there a way to speed up the convergence?

BIBLIOGRAPHY

[1] CONOPT. URL http://www.conopt.com/.

[2] Quintiq. URL http://www.quintiq.com/.

[3] LocalSolver. URL http://www.localsolver.com/.

[4] T.E. Baker and L.S. Lasdon. Successive linear programming at Exxon. *Management Science*, 1985.

[5] J. Bengtsson and S.L. Nonås. Refinery planning and scheduling - an overview. *SNF Report No. 29/08*, 2008.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] K. Glissmann and G. Gruhn. Short-term scheduling and recipe optimization of blending processes. *Elsevier*, 2001.

[8] Z. Jia and M. Ierapetritou. Mixed-integer linear programming model for gasoline blending and distribution scheduling. *Ind. Eng. Chem. Res.*, 2003.

[9] R. Karuppiah, K.C.. Furman, and I.E. Grossmann. Global optimization for scheduling refinery crude oil operations. *Computers and Chemical Engineering 32 (11)*, 2007.

[10] J.D. Kelly and J.L. Mann. Crude-Oil Blend Scheduling Optimization: An Application with Multi-Million Dollar Benefits. *Hydrocarbon processing*, 2003.

[11] H. Lee, J.M. Pinto, I.E. Grossmann, and S. Park. Mixed-integer linear programming model for refinery short-term scheduling of crude oil unloading with inventory management. *American Chemical Society*, 1996.

[12] A. Mitsos, B. Chachuat, and P.I. Barton. McCormick-based relaxations of algorithms. *Society for Industrial and Applied Mathematics*, 2009.

[13] C. A. Méndez, I. E. Grossmann, I. Harjunkoski, and P. Kaboré. A simultaneous optimization approach for off-line blending and scheduling of oil-refinery operations. *Computers and Chemical Engineering 30*, pages 614–634, 2006b.

[14] S. Mouret. COSP Model using MOS, 2010. URL http://minlp.org/library/problem/mod/index.php?lib=MINLP&i=149&ver=152&pi=117.

[15] S. Mouret. *Optimal Scheduling of Refinery Crude-Oil Operations*. PhD thesis, Carnegie Mellon University, 2010.

Part IV

Appendix

11

CP MODEL

This appendix presents the proposed CP model in OPL code.

```
using CP;

tuple Vessel {
  key string vesselId;
  int arrivalTime;
  string crudeId;
  int initAmount;
  string type;
}

{Vessel} Vessels = ...;

tuple StorageTank {
  key string storageTankId;
  int mincapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
}

{StorageTank} StorageTanks = ...;

tuple ChargingTank {
  key string chargingTankId;
  int mincapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
  string crudeMixId;
}

{ChargingTank} ChargingTanks = ...;

tuple Clique {
  key int first;
  key int second;
  key int third;
}

{Clique} Cliques = ...;

tuple Unit {
  key string unitId;
}
```

```
{Unit} Units = ...;

tuple Resource {
  key string resourceId;
  string resourceType;
  int minCapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
}

{Resource} Resources = ...;

int Horizon = ...;

int Distillation = ...;

int Time = ...;

int Lowerbound = ...;

int Upperbound = ...;

tuple CrudeMix {
  key string crudeMixId;
  float prop1Lower;
  float prop1Upper;
  float prop2Lower;
  float prop2Upper;
  float prop3Lower;
  float prop3Upper;
  int demandLower;
  int demandUpper;
}

{CrudeMix} CrudeMixes = ...;

tuple Crude {
  key string crudeId;
  float prop1;
  float prop2;
  float prop3;
  float grossMargin;
}

{Crude} Crudes = ...;

tuple PrioritySlot {
  key int slotId;
}
```

```
{PrioritySlot} PrioritySlots = ...;

tuple Operation {
  key int operationId;
  string fromResourceId;
  string toResourceId;
  int minFlowrate;
  int maxFlowrate;
  string operationType;
  string vesselType;
}

{Operation} Operations = ...;

tuple AssignedSlotOperation {
  key int prioritySlotId;
  key int operationId;
}

{AssignedSlotOperation} AssignedSlotOperations = ...;

tuple OperationCrude {
  key int operationId;
  key string crudeId;
}

{OperationCrude} OperationCrudes = ...;

tuple SlotOperation {
  key PrioritySlot prioritySlot;
  key Operation operation;
}

{SlotOperation} SlotOperations = {<s,o> | s in PrioritySlots, o
    in Operations, so in AssignedSlotOperations:
                        s.slotId==so.prioritySlotId && o.
                            operationId==so.operationId };

tuple SlotOperationCrude {
  key PrioritySlot prioritySlot;
  key Operation operation;
  key Crude crude;
}

{SlotOperationCrude} SlotOperationCrudes = {<s,o,c> | <s,o> in
    SlotOperations, c in Crudes, oc in OperationCrudes:
                                oc.operationId==o.operationId &&
                                    oc.crudeId==c.crudeId};

tuple SlotResource {
  key PrioritySlot prioritySlot;
  key Resource resource;
```

```
}

{SlotResource} SlotResources = {<p,r> | p in PrioritySlots, r in
    Resources};

tuple SlotResourceCrude {
  key PrioritySlot prioritySlot;
  key Resource resource;
  key Crude crude;
}

{SlotResourceCrude} SlotResourceCrudes = {<p,r,c> | p in
    PrioritySlots, r in Resources, c in Crudes, o in Operations,
    oc in OperationCrudes:
          r.resourceId==o.fromResourceId && oc.operationId==o.
              operationId && oc.crudeId==c.crudeId};

tuple SlotTank {
  key PrioritySlot prioritySlot;
  key Resource resource;
}

{SlotTank} SlotTanks = {<p,r> | p in PrioritySlots, r in
    Resources: r.resourceType=="Tank"};

tuple SlotTankCrude {
  key PrioritySlot prioritySlot;
  key Resource resource;
  key Crude crude;
}

{SlotTankCrude} SlotTankCrudes = {<p,r,c> | <p,r> in SlotTanks, c
     in Crudes, o in Operations, oc in OperationCrudes:
          r.resourceId==o.fromResourceId && oc.operationId==o.
              operationId && oc.crudeId==c.crudeId};

tuple ResourceCrude {
  key Resource resource;
  key Crude crude;
}

{ResourceCrude} ResourceCrudes = {<r,c> | r in Resources, o in
    Operations, oc in OperationCrudes, c in Crudes
      : r.resourceType=="Tank" && o.operationId==oc.operationId
        && oc.crudeId==c.crudeId && o.fromResourceId==r.
            resourceId};


int OverlapMatrix[Operations][Operations];
execute {
  for(var i in Operations)
  {
```

```
    for(var j in Operations)
    {
      if( ( i.operationId==j.operationId ) //self
         || ( ( i.operationType=="Unloading"
            && j.operationType=="Unloading"
            && i.vesselType==j.vesselType )
            && ( i.operationId != j.operationId ) ) //vessels
         || ( i.toResourceId==j.fromResourceId
            || i.fromResourceId==j.toResourceId ) //inflow
                outflow
         || ( i.fromResourceId==j.fromResourceId
            && ( i.operationId != j.operationId )
            && i.operationType=="Distillation" ) //double out
                charging tanks
         || ( i.toResourceId==j.toResourceId
              && ( i.operationId != j.operationId )
            && i.operationType=="Distillation" ) //double in CDUs
          )
      {
      OverlapMatrix[i][j] = 0;
      }
      else
      {
        OverlapMatrix[i][j] = 1;
      }
    }
  }
}

int quantity = 1;

range volumeRange = 0..9999*quantity;
range cvolumeRange = 0..9999*quantity;
range levelRange = 0..9999*quantity;
range timeRange = 0..Horizon;

dvar int start[so in SlotOperations] in timeRange;

dvar int duration[so in SlotOperations] in timeRange;

dvar int end[so in SlotOperations] in timeRange;

dvar int tvolume[so in SlotOperations] in volumeRange;

dvar int cvolume[soc in SlotOperationCrudes] in cvolumeRange;

dvar int tlevel[t in SlotTanks] in levelRange;

dvar int clevel[tc in SlotTankCrudes] in levelRange;
```

```
dexpr float TotalGrossMargin = sum(<s,o,c> in SlotOperationCrudes
    : o.operationType=="Distillation")(c.grossMargin*cvolume[<s,o
    ,c>])/quantity;

execute {
  cp.param.Workers = 1;
    cp.param.TimeLimit = 60;

    var f = cp.factory;

    cp.setSearchPhases(f.searchPhase(cvolume), f.searchPhase(
        start) );
}

maximize
  TotalGrossMargin;
subject to {

forall(<s,o> in SlotOperations)
    start[<s,o>] + duration[<s,o>] == end[<s,o>];

forall(<s,o> in SlotOperations, v in Vessels: o.operationType=="
    Unloading" && v.vesselId==o.fromResourceId)
    start[<s,o>] >= v.arrivalTime;

forall(<s,o> in SlotOperations, v in Vessels: o.operationType=="
    Unloading" && v.vesselId==o.fromResourceId)
    tvolume[<s,o>] == v.initAmount*quantity;

forall(<s,o> in SlotOperations, r in Resources: o.operationType
    !="Unloading" && r.resourceId==o.fromResourceId)
    tvolume[<s,o>] <= r.maxCapacity*quantity;

forall(<s,o> in SlotOperations, <s2,o2> in SlotOperations:
      s.slotId+1==s2.slotId && OverlapMatrix[o][o2]==0)
    end[<s,o>] <= start[<s2,o2>];

forall( v in Vessels, v2 in Vessels: v.arrivalTime < v2.
    arrivalTime && v.type == v2.type)
    sum(<s,o> in SlotOperations: o.fromResourceId == v.vesselId)
        end[<s,o>]
      <= sum(<s,o> in SlotOperations: o.fromResourceId == v2.
          vesselId)start[<s,o>];

forall(u in Units)
    Horizon==sum(<s,o> in SlotOperations: o.toResourceId==u.
        unitId)duration[<s,o>];

forall(<s,o> in SlotOperations)
      tvolume[<s,o>]==sum(<s,o,c> in SlotOperationCrudes)cvolume
          [<s,o,c>];
```

```
forall(<s,o> in SlotOperations: o.operationType=="Distillation")
       o.minFlowrate*quantity*duration[<s,o>] <= tvolume[<s,o>]*
           Time
       && o.maxFlowrate*quantity*duration[<s,o>] >= tvolume[<s,o
           >]*Time;

forall(<s,o> in SlotOperations: o.operationType!="Distillation")
       o.maxFlowrate*quantity*duration[<s,o>] == tvolume[<s,o>]*
           Time;

forall(<s,o> in SlotOperations, ct in ChargingTanks, m in
    CrudeMixes:
      ct.crudeMixId==m.crudeMixId && o.fromResourceId==ct.
          chargingTankId)
      ( m.prop1Lower*tvolume[<s,o>] <= sum(<s,o,c> in
          SlotOperationCrudes)c.prop1*cvolume[<s,o,c>] )
    && ( m.prop1Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop1*cvolume[<s,o,c>] )
    && ( m.prop2Lower*tvolume[<s,o>] <= sum(<s,o,c> in
        SlotOperationCrudes)c.prop2*cvolume[<s,o,c>] )
    && ( m.prop2Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop2*cvolume[<s,o,c>] )
    && ( m.prop3Lower*tvolume[<s,o>] <= sum(<s,o,c> in
        SlotOperationCrudes)c.prop3*cvolume[<s,o,c>] )
    && ( m.prop3Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop3*cvolume[<s,o,c>] )
    ;

forall(<s,r> in SlotResources, <s,o> in SlotOperations, c in
    Crudes, oc in OperationCrudes:
          r.resourceType=="Tank" && oc.operationId==o.operationId
               && oc.crudeId==c.crudeId && o.fromResourceId==r.
              resourceId)
      cvolume[<s,o,c>] * tlevel[<s,r>] == tvolume[<s,o>] * clevel
          [<s,r,c>];

forall(ct in ChargingTanks, m in CrudeMixes: ct.crudeMixId==m.
    crudeMixId)
      m.demandLower*quantity <= sum(<s,o> in SlotOperations: o.
          fromResourceId==ct.chargingTankId)tvolume[<s,o>]
      && m.demandUpper*quantity >= sum(<s,o> in SlotOperations: o
          .fromResourceId==ct.chargingTankId)tvolume[<s,o>];

forall(<p,r> in SlotResources: r.resourceType=="Tank" )
      tlevel[<p,r>] == quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              s.slotId<p.slotId && o.toResourceId==r.resourceId)(
                  tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
             s.slotId<p.slotId && o.fromResourceId==r.resourceId)(
                 tvolume[<s,o>]);
```

```
forall(<p,r,c> in SlotResourceCrudes: r.resourceType=="Tank" )
      clevel[<p,r,c>] == quantity*r.initAmount*(r.crudeId==c.
          crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              s.slotId<p.slotId && o.toResourceId==r.resourceId)(
                  cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              s.slotId<p.slotId && o.fromResourceId==r.resourceId
                  )(cvolume[<s,o,c>]);
            ;
forall(<p,r> in SlotResources: r.resourceType=="Tank")
      tlevel[<p,r>]==sum(<p,r,c> in SlotResourceCrudes)clevel[<p,
          r,c>];

forall(<p,r> in SlotResources: r.resourceType=="Tank")
       r.minCapacity*quantity<=tlevel[<p,r>] && tlevel[<p,r>]<=
           quantity*r.maxCapacity;

forall(<p,r,c> in SlotResourceCrudes: r.resourceType=="Tank")
       0 <= clevel[<p,r,c>] && clevel[<p,r,c>] <= r.maxCapacity*
           quantity;

forall(r in Resources: r.resourceType=="Tank" )
      r.minCapacity*quantity <= quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              o.toResourceId==r.resourceId)(tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
              o.fromResourceId==r.resourceId)(tvolume[<s,o>]);

forall(r in Resources: r.resourceType=="Tank" )
      r.maxCapacity*quantity >= quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              o.toResourceId==r.resourceId)(tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
              o.fromResourceId==r.resourceId)(tvolume[<s,o>]);

forall(<r,c> in ResourceCrudes: r.resourceType=="Tank")
      0 <= r.initAmount*quantity*(r.crudeId==c.crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              o.toResourceId==r.resourceId)(cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              o.fromResourceId==r.resourceId)(cvolume[<s,o,c>]);

forall(<r,c> in ResourceCrudes: r.resourceType=="Tank")
      r.maxCapacity*quantity >= r.initAmount*quantity*(r.crudeId
          ==c.crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              o.toResourceId==r.resourceId)(cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              o.fromResourceId==r.resourceId)(cvolume[<s,o,c>]);

TotalGrossMargin <= Upperbound;
```

```
}

tuple Output {
  key int slot;
  key int operation;
  float duration;
  int volume;
};

{Output} Outputs = {<s.slotId, o.operationId, duration[<s,o>]/
    Time, tvolume[<s,o>]> | <s,o> in SlotOperations };

execute {
  writeln("Total Gross Margin: ", TotalGrossMargin);
  writeln();

  for(var o in Outputs)
  {   writeln(o.slot, ", ",
          o.operation, ": D: ", o.duration, ", V: ", o.volume);
  }
  writeln();
}
```

12

CP MODEL VARIANT

This appendix presents the proposed variant on the CP model in OPL code.

```
using CP;

tuple Vessel {
  key string vesselId;
  int arrivalTime;
  string crudeId;
  int initAmount;
  string type;
}

{Vessel} Vessels = ...;

tuple StorageTank {
  key string storageTankId;
  int mincapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
}

{StorageTank} StorageTanks = ...;

tuple ChargingTank {
  key string chargingTankId;
  int mincapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
  string crudeMixId;
}

{ChargingTank} ChargingTanks = ...;

tuple Unit {
  key string unitId;
}

tuple Clique {
  key int first;
  key int second;
  key int third;
}
```

```
{Clique} Cliques = ...;

{Unit} Units = ...;

tuple Resource {
  key string resourceId;
  string resourceType;
  int minCapacity;
  int maxCapacity;
  string crudeId;
  int initAmount;
}

{Resource} Resources = ...;

int Horizon = ...;

int Distillation = ...;

int Time = ...;

int Lowerbound = ...;

int Upperbound = ...;

tuple CrudeMix {
  key string crudeMixId;
  float prop1Lower;
  float prop1Upper;
  float prop2Lower;
  float prop2Upper;
  float prop3Lower;
  float prop3Upper;
  int demandLower;
  int demandUpper;
}

{CrudeMix} CrudeMixes = ...;

tuple Crude {
  key string crudeId;
  float prop1;
  float prop2;
  float prop3;
  float grossMargin;
}

{Crude} Crudes = ...;

tuple PrioritySlot {
  key int slotId;
}
```

```
{PrioritySlot} PrioritySlots = ...;

tuple Operation {
  key int operationId;
  string fromResourceId;
  string toResourceId;
  int minFlowrate;
  int maxFlowrate;
  string operationType;
  string vesselType;
}

{Operation} Operations = ...;

tuple AssignedSlotOperation {
  key int prioritySlotId;
  key int operationId;
}

{AssignedSlotOperation} AssignedSlotOperations = ...;

tuple OperationCrude {
  key int operationId;
  key string crudeId;
}

{OperationCrude} OperationCrudes = ...;

tuple SlotOperation {
  key PrioritySlot prioritySlot;
  key Operation operation;
}

{SlotOperation} SlotOperations = {<s,o> | s in PrioritySlots, o
    in Operations };

tuple SlotOperationCrude {
  key PrioritySlot prioritySlot;
  key Operation operation;
  key Crude crude;
}

{SlotOperationCrude} SlotOperationCrudes = {<s,o,c> | <s,o> in
    SlotOperations, c in Crudes, oc in OperationCrudes:
                              oc.operationId==o.operationId &&
                                  oc.crudeId==c.crudeId};

tuple SlotResource {
  key PrioritySlot prioritySlot;
  key Resource resource;
}
```

```
{SlotResource} SlotResources = {<p,r> | p in PrioritySlots, r in
    Resources};

tuple SlotResourceCrude {
  key PrioritySlot prioritySlot;
  key Resource resource;
  key Crude crude;
}

{SlotResourceCrude} SlotResourceCrudes = {<p,r,c> | p in
    PrioritySlots, r in Resources, c in Crudes, o in Operations,
    oc in OperationCrudes:
          r.resourceId==o.fromResourceId && oc.operationId==o.
              operationId && oc.crudeId==c.crudeId};

tuple SlotTank {
  key PrioritySlot prioritySlot;
  key Resource resource;
}

{SlotTank} SlotTanks = {<p,r> | p in PrioritySlots, r in
    Resources: r.resourceType=="Tank"};

tuple SlotTankCrude {
  key PrioritySlot prioritySlot;
  key Resource resource;
  key Crude crude;
}

{SlotTankCrude} SlotTankCrudes = {<p,r,c> | <p,r> in SlotTanks, c
     in Crudes, o in Operations, oc in OperationCrudes:
          r.resourceId==o.fromResourceId && oc.operationId==o.
              operationId && oc.crudeId==c.crudeId};

tuple ResourceCrude {
  key Resource resource;
  key Crude crude;
}

{ResourceCrude} ResourceCrudes = {<r,c> | r in Resources, o in
    Operations, oc in OperationCrudes, c in Crudes
      : r.resourceType=="Tank" && o.operationId==oc.operationId
        && oc.crudeId==c.crudeId && o.fromResourceId==r.
            resourceId};



int OverlapMatrix[Operations][Operations];
execute {
  for(var i in Operations)
  {
```

```
    for(var j in Operations)
    {
      if( ( i.operationId==j.operationId ) //self
         || ( ( i.operationType=="Unloading"
            && j.operationType=="Unloading"
            && i.vesselType==j.vesselType )
            && ( i.operationId != j.operationId ) ) //vessels
         || ( i.toResourceId==j.fromResourceId
            || i.fromResourceId==j.toResourceId ) //inflow
                outflow
         || ( i.fromResourceId==j.fromResourceId
            && ( i.operationId != j.operationId )
            && i.operationType=="Distillation" ) //double out
                charging tanks
         || ( i.toResourceId==j.toResourceId
              && ( i.operationId != j.operationId )
            && i.operationType=="Distillation" ) //double in CDUs
          )
      {
      OverlapMatrix[i][j] = 0;
      }
      else
      {
        OverlapMatrix[i][j] = 1;
      }
    }
  }
}

int quantity = 1;

range volumeRange = 0..9999*quantity;
range cvolumeRange = 0..9999*quantity;
range levelRange = 0..9999*quantity;
range timeRange = 0..Horizon;

dvar boolean assign[so in SlotOperations];

dvar int start[so in SlotOperations] in timeRange;

dvar int duration[so in SlotOperations] in timeRange;

dvar int end[so in SlotOperations] in timeRange;

dvar int tvolume[so in SlotOperations] in volumeRange;

dvar int cvolume[soc in SlotOperationCrudes] in cvolumeRange;

dvar int tlevel[t in SlotTanks] in levelRange;

dvar int clevel[tc in SlotTankCrudes] in levelRange;
```

```
dexpr float TotalGrossMargin = sum(<s,o,c> in SlotOperationCrudes
    : o.operationType=="Distillation")(c.grossMargin*cvolume[<s,o
    ,c>])/quantity;

execute {
  cp.param.Workers = 1;
    cp.param.TimeLimit = 60;

    var f = cp.factory;

    cp.setSearchPhases(f.searchPhase(assign), f.searchPhase(
        cvolume), f.searchPhase(start) );
}

maximize
  TotalGrossMargin;
subject to {

forall(<s,o> in SlotOperations, v in Vessels: o.operationType=="
    Unloading" && v.vesselId==o.fromResourceId)
    start[<s,o>] >= v.arrivalTime * assign[<s,o>];

forall(<s,o> in SlotOperations)
    end[<s,o>] <= Horizon * assign[<s,o>];

forall(<s,o> in SlotOperations)
    start[<s,o>] + duration[<s,o>] == end[<s,o>];

Distillation == sum(<s,o> in SlotOperations: o.operationType == "
    Distillation")assign[<s,o>];

forall(<s,o> in SlotOperations, <s,o2> in SlotOperations:
    OverlapMatrix[o][o2]==0 && o.operationId < o2.operationId)
      (assign[<s,o>] + assign[<s,o2>]) <= 1;

forall(<s,o> in SlotOperations, <s2,o> in SlotOperations:
      s.slotId<s2.slotId && OverlapMatrix[o][o]==0)
    end[<s,o>] <= start[<s2,o>] + Horizon * (1-assign[<s2,o>]);

forall(<s,o> in SlotOperations, <s,o2> in SlotOperations, <s2,o>
    in SlotOperations, <s2,o2> in SlotOperations:
      s.slotId<s2.slotId && o.operationId != o2.operationId &&
          OverlapMatrix[o][o2]==0)
    end[<s,o>] + end[<s,o2>] <= start[<s2,o>] + start[<s2,o2>] +
        Horizon * (1-assign[<s2,o>]-assign[<s2,o2>]);

forall( v in Vessels, v2 in Vessels: v.arrivalTime < v2.
    arrivalTime && v.type == v2.type)
    sum(<s,o> in SlotOperations: o.fromResourceId == v.vesselId)
        end[<s,o>]
      <= sum(<s,o> in SlotOperations: o.fromResourceId == v2.
          vesselId)start[<s,o>];
```

```
forall( s in PrioritySlots, v in Vessels, v2 in Vessels: v.
    arrivalTime < v2.arrivalTime && v.type == v2.type)
    (sum(<s2,o> in SlotOperations: o.fromResourceId == v.vesselId
         && s2.slotId < s.slotId)(assign[<s2,o>]))
      >= (sum(<s2,o> in SlotOperations: o.fromResourceId == v2.
          vesselId && s2.slotId <= s.slotId)(assign[<s2,o>]));

forall( v in Vessels)
    sum(<s,o> in SlotOperations: o.fromResourceId == v.vesselId)
        assign[<s,o>] == 1;

forall(u in Units)
    Horizon==sum(<s,o> in SlotOperations: o.toResourceId==u.
        unitId)duration[<s,o>];

forall(<s,o> in SlotOperations, v in Vessels: o.operationType=="
    Unloading" && v.vesselId==o.fromResourceId)
    tvolume[<s,o>] == v.initAmount*quantity * assign[<s,o>];

forall(<s,o> in SlotOperations, r in Resources: o.operationType
    !="Unloading" && r.resourceId==o.fromResourceId)
    tvolume[<s,o>] <= r.maxCapacity*quantity * assign[<s,o>];

forall(<s,o> in SlotOperations)
      tvolume[<s,o>]==sum(<s,o,c> in SlotOperationCrudes)cvolume
          [<s,o,c>];

forall(<s,o> in SlotOperations)
       o.minFlowrate*quantity*duration[<s,o>] <= tvolume[<s,o>]*
           Time
       && o.maxFlowrate*quantity*duration[<s,o>] >= tvolume[<s,o
           >]*Time;

forall(<s,o> in SlotOperations, ct in ChargingTanks, m in
    CrudeMixes:
      ct.crudeMixId==m.crudeMixId && o.fromResourceId==ct.
          chargingTankId)
      ( m.prop1Lower*tvolume[<s,o>] <= sum(<s,o,c> in
          SlotOperationCrudes)c.prop1*cvolume[<s,o,c>] )
    && ( m.prop1Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop1*cvolume[<s,o,c>] )
    && ( m.prop2Lower*tvolume[<s,o>] <= sum(<s,o,c> in
        SlotOperationCrudes)c.prop2*cvolume[<s,o,c>] )
    && ( m.prop2Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop2*cvolume[<s,o,c>] )
    && ( m.prop3Lower*tvolume[<s,o>] <= sum(<s,o,c> in
        SlotOperationCrudes)c.prop3*cvolume[<s,o,c>] )
    && ( m.prop3Upper*tvolume[<s,o>] >= sum(<s,o,c> in
        SlotOperationCrudes)c.prop3*cvolume[<s,o,c>] )
    ;
```

```
forall(<s,r> in SlotResources, <s,o> in SlotOperations, c in
    Crudes, oc in OperationCrudes:
          r.resourceType=="Tank" && oc.operationId==o.operationId
               && oc.crudeId==c.crudeId && o.fromResourceId==r.
              resourceId)
      cvolume[<s,o,c>] * tlevel[<s,r>] == tvolume[<s,o>] * clevel
          [<s,r,c>];

forall(ct in ChargingTanks, m in CrudeMixes: ct.crudeMixId==m.
    crudeMixId)
      m.demandLower*quantity <= sum(<s,o> in SlotOperations: o.
          fromResourceId==ct.chargingTankId)tvolume[<s,o>]
      && m.demandUpper*quantity >= sum(<s,o> in SlotOperations: o
          .fromResourceId==ct.chargingTankId)tvolume[<s,o>];

forall(<p,r> in SlotResources: r.resourceType=="Tank" )
      tlevel[<p,r>] == quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              s.slotId<p.slotId && o.toResourceId==r.resourceId)(
                  tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
             s.slotId<p.slotId && o.fromResourceId==r.resourceId)(
                 tvolume[<s,o>]);

forall(<p,r,c> in SlotResourceCrudes: r.resourceType=="Tank" )
      clevel[<p,r,c>] == quantity*r.initAmount*(r.crudeId==c.
          crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              s.slotId<p.slotId && o.toResourceId==r.resourceId)(
                  cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              s.slotId<p.slotId && o.fromResourceId==r.resourceId
                  )(cvolume[<s,o,c>]);
            ;
forall(<p,r> in SlotResources: r.resourceType=="Tank")
      tlevel[<p,r>]==sum(<p,r,c> in SlotResourceCrudes)clevel[<p,
          r,c>];

forall(<p,r> in SlotResources: r.resourceType=="Tank")
       r.minCapacity*quantity<=tlevel[<p,r>] && tlevel[<p,r>]<=
           quantity*r.maxCapacity;

forall(<p,r,c> in SlotResourceCrudes: r.resourceType=="Tank")
       0 <= clevel[<p,r,c>] && clevel[<p,r,c>] <= r.maxCapacity*
           quantity;

forall(r in Resources: r.resourceType=="Tank" )
      r.minCapacity*quantity <= quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              o.toResourceId==r.resourceId)(tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
              o.fromResourceId==r.resourceId)(tvolume[<s,o>]);
```

```
forall(r in Resources: r.resourceType=="Tank" )
      r.maxCapacity*quantity >= quantity*r.initAmount
              + sum(<s,o> in SlotOperations:
              o.toResourceId==r.resourceId)(tvolume[<s,o>])
              - sum(<s,o> in SlotOperations:
              o.fromResourceId==r.resourceId)(tvolume[<s,o>]);

forall(<r,c> in ResourceCrudes: r.resourceType=="Tank")
      0 <= r.initAmount*quantity*(r.crudeId==c.crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              o.toResourceId==r.resourceId)(cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              o.fromResourceId==r.resourceId)(cvolume[<s,o,c>]);

forall(<r,c> in ResourceCrudes: r.resourceType=="Tank")
      r.maxCapacity*quantity >= r.initAmount*quantity*(r.crudeId
          ==c.crudeId)
              + sum(<s,o,c> in SlotOperationCrudes:
              o.toResourceId==r.resourceId)(cvolume[<s,o,c>])
              - sum(<s,o,c> in SlotOperationCrudes:
              o.fromResourceId==r.resourceId)(cvolume[<s,o,c>]);

forall(<s,o> in SlotOperations, z in AssignedSlotOperations: z.
    prioritySlotId == s.slotId && o.operationId == z.operationId
    )
      assign[<s,o>] == 1;

TotalGrossMargin <= Upperbound;
}

tuple Output {
  key int slot;
  key int operation;
  float duration;
  int volume;
};

{Output} Outputs = {<s.slotId, o.operationId, duration[<s,o>]/
    Time, tvolume[<s,o>]> | <s,o> in SlotOperations };

execute {
  writeln("Total Gross Margin: ", TotalGrossMargin);
  writeln();

  for(var o in Outputs)
  {   writeln(o.slot, ", ",
          o.operation, ": D: ", o.duration, ", V: ", o.volume);
  }
  writeln();
}
```

13 LOCALSOLVER MODEL

This appendix presents the LocalSolver model for COSP1.

```
function input(){
  H=8;
  PS=5;

  //volume at time t of element j  of oil p
  //vessel 1
  L[0][1][1] = 1000;
  L[0][1][2..4] = 0;

  //vessel 2
  L[0][2][1] = 0;
  L[0][2][2] = 1000;
  L[0][2][3..4] = 0;

  //storage tank 1
  L[0][3][1] = 250;
  L[0][3][2..4] = 0;

  //storage tank 2
  L[0][4][1] = 0;
  L[0][4][2] = 750;
  L[0][4][3..4] = 0;

  //charging tank 1
  L[0][5][1..2] = 0;
  L[0][5][3] = 500;
  L[0][5][4] = 0;

  //charging tank 2
  L[0][6][1..3] = 0;
  L[0][6][4] = 500;

  //CDU
  L[0][7][1..4] = 0;

  //sulfure:
  C[1] = 0.01;
  C[2] = 0.06;
  C[3] = 0.02;
  C[4] = 0.05;

  //margin by Mbbl
  MARGIN[1] = 1;
  MARGIN[2] = 6;
```

```
  MARGIN[3] = 2;
  MARGIN[4] = 5;

  LABELS[1] = "V1";
  LABELS[2] = "V2";
  LABELS[3] = "ST1";
  LABELS[4] = "ST2";
  LABELS[5] = "CT1";
  LABELS[6] = "CT2";
  LABELS[7] = "CDU";

  //        0,1,2,3,4,5,6,7,8
  overlap[1] = {0,0,0,0,0,1,1,1,1};
  overlap[2] = {0,0,0,1,1,0,0,1,1};
  overlap[3] = {0,0,1,0,1,1,1,0,1};
  overlap[4] = {0,0,1,1,0,1,1,1,0};
  overlap[5] = {0,1,0,1,1,0,1,0,1};
  overlap[6] = {0,1,0,1,1,1,0,1,0};
  overlap[7] = {0,1,1,0,1,0,1,0,0};
  overlap[8] = {0,1,1,1,0,1,0,0,0};
}

function model(){
  z[1..PS][1..8] <- bool();
  s[1..PS][1..8] <- float(0,H);
  e[1..PS][1..8] <- float(0,H);

  constraint z[1][1] ==0;
  constraint z[1][2] ==0;
  constraint z[1][3] ==0;
  constraint z[1][4] ==0;
  constraint z[1][5] ==0;
  constraint z[1][6] ==1;
  constraint z[1][7] ==1;
  constraint z[1][8] ==0;
  constraint z[2][1] ==0;
  constraint z[2][2] ==0;
  constraint z[2][3] ==1;
  constraint z[2][4] ==0;
  constraint z[2][5] ==1;
  constraint z[2][6] ==0;
  constraint z[2][7] ==0;
  constraint z[2][8] ==1;
  constraint z[3][1] ==1;
  constraint z[3][2] ==0;
  constraint z[3][3] ==0;
  constraint z[3][4] ==0;
  constraint z[3][5] ==0;
  constraint z[3][6] ==1;
  constraint z[3][7] ==0;
  constraint z[3][8] ==0;
  constraint z[4][1] ==0;
```

```
constraint z[4][2] ==1;
constraint z[4][3] ==1;
constraint z[4][4] ==0;
constraint z[4][5] ==0;
constraint z[4][6] ==0;
constraint z[4][7] ==0;
constraint z[4][8] ==0;
constraint z[5][1] ==0;
constraint z[5][2] ==0;
constraint z[5][3] ==0;
constraint z[5][4] ==0;
constraint z[5][5] ==0;
constraint z[5][6] ==0;
constraint z[5][7] ==1;
constraint z[5][8] ==0;

for[p in 1..PS][o in 1..8]{
  d[p][o] <- e[p][o] - s[p][o];
}

for[p in 1..PS][o in 1..8]{
  constraint s[p][o] <= e[p][o];
  constraint d[p][o] <= e[p][o] * z[p][o];
}

for[p in 1..PS]{
  constraint s[p][2] >= 4 * z[p][2];
}

for[p in 1..PS][o in 1..8]{
  constraint e[p][o] <= H * z[p][o];
}

constraint sum[p in 1..PS] (z[p][7] + z[p][8]) == 3;

for[p in 1..PS][o in 1..7][o2 in o+1..8: overlap[o][o2] == 0]{
  constraint z[p][o] + z[p][o2] <= 1;
}

for[p in 1..PS-1][p2 in p+1..PS][o in 1..8: overlap[o][o] ==
    0]{
  constraint e[p][o] <= s[p2][o] + H * (1-z[p2][o]) ;
}

for[p in 1..PS-1][p2 in p+1..PS][o in 1..8][o2 in 1..8: o != o2
     && overlap[o][o2] == 0]{
  constraint e[p][o] + e[p][o2] <= s[p2][o] + s[p2][o2] + H *
      (1-z[p2][o]-z[p2][o2]) ;
}

constraint sum[p in 1..PS](z[p][1]) == 1;
constraint sum[p in 1..PS](z[p][2]) == 1;
```

```
constraint sum[p in 1..PS][o in 7..8](d[p][o]) == H;

q[1..PS][1..8] <- float(0,1);

for[p in 1..PS]{
  constraint q[p][3] + q[p][4] <= 1;
  constraint q[p][5] + q[p][6] <= 1;
}

for[p in 1..PS][c in 1..4]{
  L[p][1][c] <- L[p-1][1][c] - q[p][1] * L[p-1][1][c] * z[p
      ][1];  //V1
  L[p][2][c] <- L[p-1][2][c] - q[p][2] * L[p-1][2][c] * z[p
      ][2];  //V2
  L[p][3][c] <- L[p-1][3][c] + q[p][1] * L[p-1][1][c] * z[p][1]
       - q[p][3] * L[p-1][3][c] * z[p][3] - q[p][4] * L[p
      -1][3][c] * z[p][4]; //ST1
  L[p][4][c] <- L[p-1][4][c] + q[p][2] * L[p-1][2][c] * z[p][2]
       - q[p][5] * L[p-1][4][c] * z[p][5] - q[p][6] * L[p
      -1][4][c] * z[p][6]; //ST2
  L[p][5][c] <- L[p-1][5][c] + q[p][3] * L[p-1][3][c] * z[p][3]
       + q[p][5] * L[p-1][4][c] * z[p][5] - q[p][7] * L[p
      -1][5][c] * z[p][7]; //CT1
  L[p][6][c] <- L[p-1][6][c] + q[p][4] * L[p-1][3][c] * z[p][4]
       + q[p][6] * L[p-1][4][c] * z[p][6] - q[p][8] * L[p
      -1][6][c] * z[p][8]; //CT2
  L[p][7][c] <- L[p-1][7][c] + q[p][7] * L[p-1][5][c] * z[p][7]
       + q[p][8] * L[p-1][6][c] * z[p][8]; //CDU
}

for[p in 1..PS][r in 3..6][c in 1..4]{
  constraint L[p][r][c] <= 1000;
  constraint L[p][r][c] >= 0;
}

for[p in 1..PS][c in 1..4]{
  V[p][1][c] <- q[p][1] * L[p-1][1][c];
  V[p][2][c] <- q[p][2] * L[p-1][2][c];
  V[p][3][c] <- q[p][3] * L[p-1][3][c];
  V[p][4][c] <- q[p][4] * L[p-1][3][c];
  V[p][5][c] <- q[p][5] * L[p-1][4][c];
  V[p][6][c] <- q[p][6] * L[p-1][4][c];
  V[p][7][c] <- q[p][7] * L[p-1][5][c];
  V[p][8][c] <- q[p][8] * L[p-1][6][c];
}

for[p in 1..PS][r in 3..6]{
  L[p][r] <- sum[c in 1..4](L[p][r][c]);
  constraint L[p][r] <= 1000;
  constraint L[p][r] >= 0;
}
```

```
  constraint sum[c in 1..4] (L[PS][1][c]) == 0;
  constraint sum[c in 1..4] (L[PS][2][c]) == 0;

  for[p in 1..PS][o in 1..8]{
    Vt[p][o] <- sum[c in 1..4](V[p][o][c]);
  }

  for[p in 1..PS]{
    constraint 0.015 * Vt[p][7] <= sum[c in 1..4](V[p][7][c] * C[
        c]);
    constraint sum[c in 1..4](V[p][7][c] * C[c]) <= 0.025 * Vt[p
        ][7];
    constraint 0.045 * Vt[p][8] <= sum[c in 1..4](V[p][8][c] * C[
        c]);
    constraint sum[c in 1..4](V[p][8][c] * C[c]) <= 0.055 * Vt[p
        ][8];
  }

  for[o in 7..8]{
    constraint sum[p in 1..PS] (Vt[p][o]*z[p][o]) == 1000;
  }

  for[p in 1..PS][o in 1..6]{
    constraint 0 * d[p][o] <= Vt[p][o];
    constraint Vt[p][o] == 500 * d[p][o];
  }

  for[p in 1..PS][o in 7..8]{
    constraint 50 * d[p][o] <= Vt[p][o];
    constraint Vt[p][o] <= 500 * d[p][o];
  }

  maximize sum[c in 1..4] (L[PS][7][c] * MARGIN[c]);
}

function param(){
  lsSeed=1;
  lsTimeLimit={10};
  setObjectiveBound(0,20000);
  lsTimeBetweenDisplays = 1;
  lsNbThreads = 2;
}

function output() {
  solFile = openWrite("solution");
  for[p in 1..PS][o in 1..8]{
  if (getValue(z[p][o]) == 1){
    println(solFile, "assign: ", p, " ", o, " assign: ", getValue
        (z[p][o]), " s: ", getValue(s[p][o]), " d: ", getValue(d[
        p][o]), " e: ",getValue(e[p][o]), " volume: ", getValue(
        Vt[p][o]), " ");
```

```
    }
  }
  println(solFile);
}
```

14 PROBLEM INSTANCES OF THE COSP

This chapter presents the remaining problem instances that were used to test the optimizers.

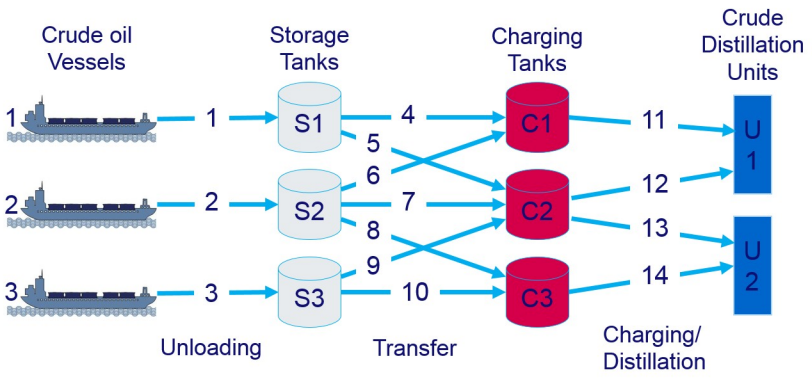

Figure 23: Crude oil operations network for COSP2 and COSP3 from Lee et al. 1996 [11]

Table 21: Overview of the COSP2 data

Scheduling horizon			10 days
Vessels	Arrival time	Composition	Amount of crude (Mbbl)
Vessel 1	0	100% A	1,000
Vessel 2	3	100% B	1,000
Vessel 3	6	100% C	1,000
Storage tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 1000]	100% A	200
Tank 2	[0, 1000]	100% B	500
Tank 3	[0, 1000]	100% C	700
Charging tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]	100% D	300
Tank 2 (mix Y)	[0, 1000]	100% E	500
Tank 3 (mix Z)	[0, 1000]	100% F	300
Crudes	Property 1	Property 2	Gross margin ($/bbl)
Crude A	0.01	0.04	1
Crude B	0.03	0.02	3
Crude C	0.05	0.01	5
Crude D	0.0167	0.0333	1.67
Crude E	0.03	0.023	3
Crude F	0.0433	0.0133	4.33
Crude mixtures	Property 1	Property 2	Demand (Mbbl)
Crude mix X	[0.01, 0.02]	[0.03, 0.038]	[1000, 1000]
Crude mix Y	[0.025, 0.035]	[0.018, 0.027]	[1000, 1000]
Crude mix Z	[0.04, 0.048]	[0.01, 0.018]	[1000, 1000]
Unloading flowrate	[0, 500]	Transfer flowrate	[0, 500]
Distillation flowrate	[50, 500]	Number of distillations	5

Table 22: Overview of the COSP3 data

Scheduling horizon			12 days
Vessels	Arrival time	Composition	Amount of crude (Mbbl)
Vessel 1	0	100% A	500
Vessel 2	4	100% B	500
Vessel 3	8	100% C	500
Storage tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 1000]	100% D	200
Tank 2	[0, 1000]	100% E	200
Tank 3	[0, 1000]	100% F	200
Charging tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]	100% G	300
Tank 2 (mix Y)	[0, 1000]	100% E	500
Tank 3 (mix Z)	[0, 1000]	100% F	300
Crudes	Property 1		Gross margin ($/bbl)
Crude A	0.01		1
Crude B	0.085		6
Crude C	0.06		8.5
Crude D	0.02		2
Crude E	0.05		5
Crude F	0.08		8
Crude G	0.03		3
Crude mixtures	Property 1		Demand (Mbbl)
Crude mix X	[0.025, 0.035]		[500, 500]
Crude mix Y	[0.045, 0.065]		[500, 500]
Crude mix Z	[0.075, 0.085]		[500, 500]
Unloading flowrate	[0, 500]	Transfer flowrate	[0, 500]
Distillation flowrate	[50, 500]	Number of distillations	5

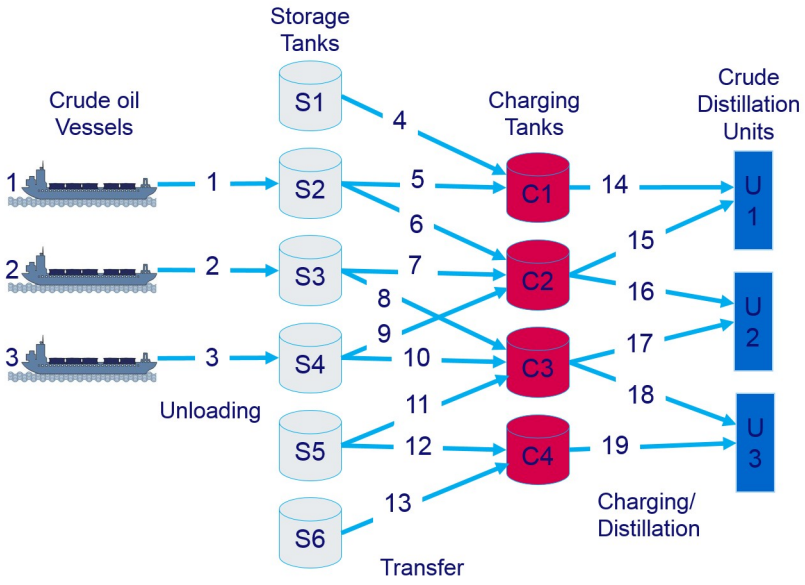

Figure 24: Crude oil operations network for COSP4 from Lee et al. 1996 [11]

Table 23: Overview of the COSP4 data

Scheduling horizon			15 days
Vessels	Arrival time	Composition	Amount of crude (Mbbl)
Vessel 1	0	100% A	600
Vessel 2	5	100% B	600
Vessel 3	10	100% C	600
Storage tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1	[100, 900]	100% D	600
Tank 2	[100, 1,100]	100% A	100
Tank 3	[100, 1,100]	100% B	500
Tank 4	[100, 1,100]	100% C	400
Tank 5	[100, 900]	100% E	300
Tank 6	[100, 900]	100% E	600
Charging tanks	Capacity (Mbbl)	Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 800]	100% F	50
Tank 2 (mix Y)	[0, 800]	100% G	300
Tank 3 (mix Z)	[0, 800]	100% H	300
Tank 4 (mix W)	[0, 800]	100% E	300
Crudes	Property 1 (sulfur concentration)		Gross margin ($/bbl)
Crude A	0.03		3
Crude B	0.05		5
Crude C	0.065		6.5
Crude D	0.031		3.1
Crude E	0.075		7.5
Crude F	0.0317		3.17
Crude G	0.0483		4.83
Crude H	0.0633		6.33
Crude mixtures	Property 1 (sulfur concentration)		Demand (Mbbl)
Crude mix X	[0.03, 0.035]		[600, 600]
Crude mix Y	[0.043, 0.05]		[600, 600]
Crude mix Z	[0.06, 0.065]		[600, 600]
Crude mix W	[0.071, 0.08]		[600, 600]
Unloading flowrate	[0, 500]	Transfer flowrate	[0, 500]
Distillation flowrate	[20, 500]	Number of distillations	7

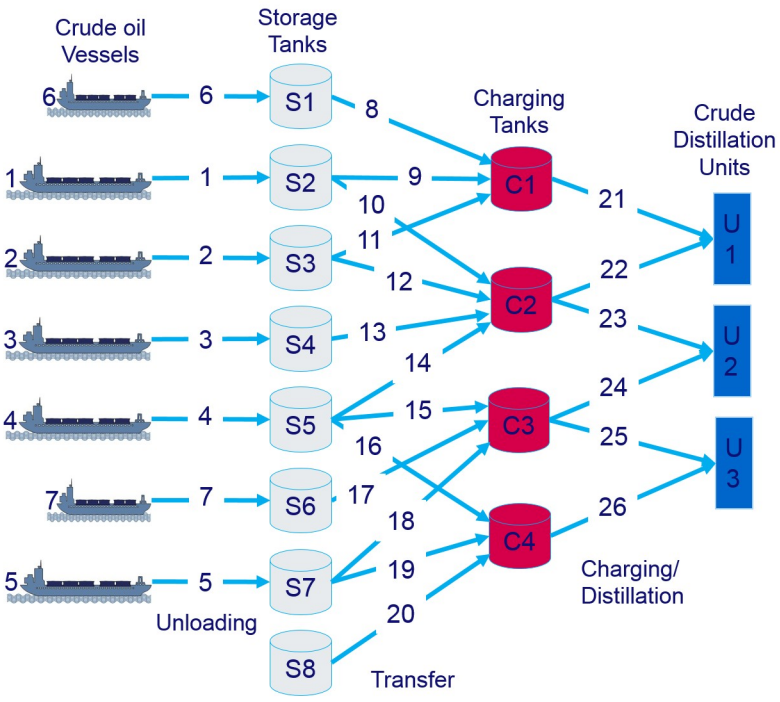

Figure 25: Crude oil operations network for COSP5

Table 24: Overview of the COSP5 data

Scheduling horizon				15
Vessels	Arrival time	Type	Composition	Amount of crude (Mbbl)
Vessel 1	0	VLCC	100% B	800
Vessel 2	10	VLCC	100% B	700
Vessel 3	4	VLCC	100% D	400
Vessel 4	5	VLCC	100% E	800
Vessel 5	7	VLCC	100% G	700
Vessel 6	7	Small	100% A	200
Vessel 7	10	Small	100% M	200
Storage tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 900]		100% A	600
Tank 2	[100, 1100]		100% B	300
Tank 3	[100, 1100]		100% B	800
Tank 4	[0, 900]		100% D	600
Tank 5	[100, 1300]		100% E	700
Tank 6	[0, 900]		100% F	800
Tank 7	[0, 1000]		100% G	400
Tank 8	[0, 900]		100% H	800
Charging tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 800]		100% I	300
Tank 2 (mix Y)	[0, 800]		100% J	400
Tank 3 (mix Z)	[0, 800]		100% K	600
Tank 4 (mix W)	[0, 800]		100% L	100
Crudes	Property 1	Property 2	Property 3	Gross margin ($/bbl)
Crude A	0.03	0.049	0.01	3.00
Crude B	0.04	0.04	0.018	4.00
Crude D	0.046	0.035	0.023	4.60
Crude E	0.05	0.03	0.026	5.00
Crude F	0.045	0.025	0.028	4.50
Crude G	0.06	0.02	0.033	6.00
Crude H	0.057	0.012	0.039	5.70
Crude I	0.0375	0.045	0.015	3.75
Crude J	0.042	0.034	0.019	4.20
Crude K	0.051	0.026	0.028	5.10
Crude L	0.057	0.013	0.038	5.70
Crude M	0.045	0.024	0.027	4.50
Crude mixtures	Property 1	Property 2	Property 3	Demand (Mbbl)
Crude mix X	[0.035, 0.039]	[0.043, 0.049]	[0.01, 0.016]	[800, 800]
Crude mix Y	[0.041, 0.045]	[0.031, 0.039]	[0.018, 0.024]	[800, 800]
Crude mix Z	[0.048, 0.054]	[0.022, 0.029]	[0.026, 0.032]	[800, 800]
Crude mix W	[0.056, 0.059]	[0.01, 0.019]	[0.034, 0.04]	[800, 800]
Unloading flowrate	[0, 500]	Transfer flowrate		[0, 500]
Distillation flowrate	[20, 500]	Number of distillation		9

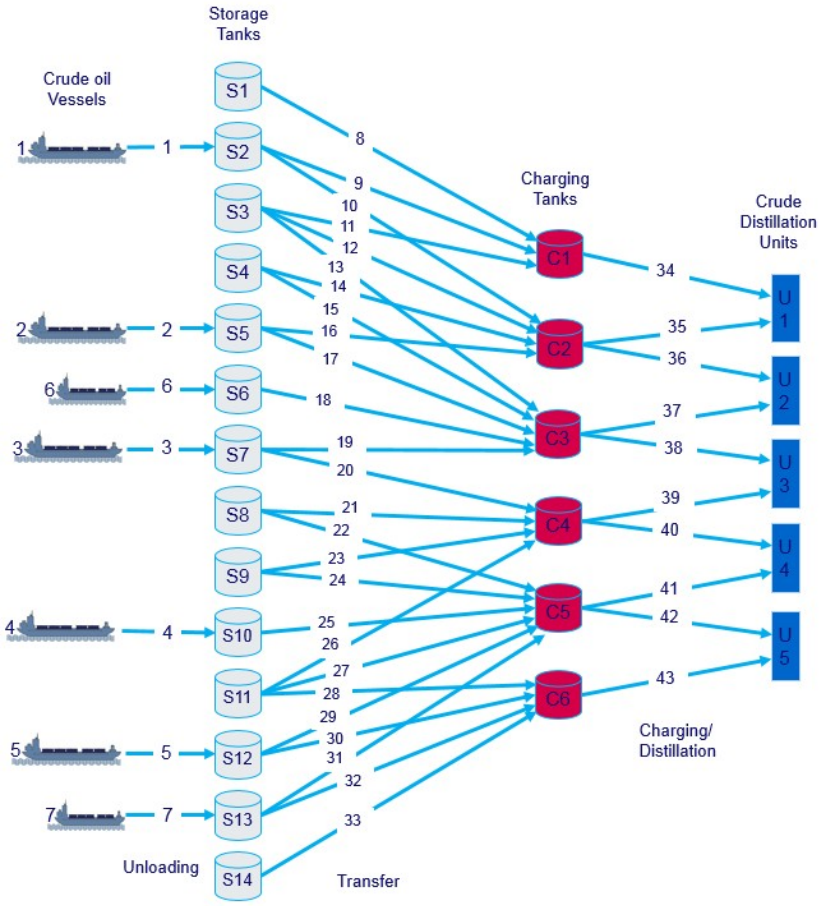

Figure 26: Crude oil operations network for COSP6

Table 25: Overview of the COSP6 data (Part 1)

Scheduling horizon				15
Vessels	Arrival time	Type	Composition	Amount of crude (Mbbl)
Vessel 1	0	VLCC	100% B	800
Vessel 2	10	VLCC	100% D	700
Vessel 3	4	VLCC	100% F	400
Vessel 4	5	VLCC	100% H	800
Vessel 5	7	VLCC	100% J	700
Vessel 6	7	Small	100% E	200
Vessel 7	10	Small	100% J	200
Storage tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 900]		100% A	600
Tank 2	[100, 1100]		100% B	300
Tank 3	[100, 1100]		100% C	800
Tank 4	[0, 900]		100% D	600
Tank 5	[100, 1300]		100% D	700
Tank 6	[0, 900]		100% E	800
Tank 7	[100, 1000]		100% F	400
Tank 8	[0, 900]		100% G	800
Tank 9	[0, 900]		100% G	600
Tank 10	[100, 1300]		100% H	700
Tank 11	[0, 900]		100% I	800
Tank 12	[0, 1000]		100% J	400
Tank 13	[0,1100]		100% J	800
Tank 14	[0, 900]		100% K	500
Charging tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]		100% L	300
Tank 2 (mix Y)	[0, 1000]		100% M	400
Tank 3 (mix Z)	[0, 1000]		100% N	600
Tank 4 (mix W)	[0, 1000]		100% O	100
Tank 5 (mix V)	[0, 1000]		100% J	500
Tank 6 (mix U)	[0, 1000]		100% P	200

Table 26: Overview of the COSP6 data (Part 2)

Crudes	Property 1	Property 2	Property 3	Gross margin ($/bbl)
Crude A	0.019	0.069	0.01	1.9
Crude B	0.03	0.06	0.015	3
Crude C	0.039	0.053	0.022	3.9
Crude D	0.042	0.048	0.027	4.2
Crude E	0.044	0.043	0.026	4.4
Crude F	0.049	0.04	0.036	4.9
Crude G	0.056	0.032	0.045	5.6
Crude H	0.059	0.027	0.052	5.9
Crude I	0.06	0.025	0.05	6
Crude J	0.058	0.022	0.053	5.8
Crude K	0.073	0.01	0.067	7.3
Crude L	0.026	0.068	0.015	2.6
Crude M	0.037	0.057	0.021	3.7
Crude N	0.044	0.048	0.032	4.4
Crude O	0.052	0.039	0.04	5.2
Crude P	0.069	0.016	0.065	6.9
Crude mixtures	Property 1	Property 2	Property 3	Demand (Mbbl)
Crude mix X	[0.020, 0.027]	[0.061, 0.069]	[0.01, 0.016]	[1000, 1000]
Crude mix Y	[0.035, 0.039]	[0.052, 0.059]	[0.018, 0.024]	[900, 900]
Crude mix Z	[0.041, 0.045]	[0.043, 0.049]	[0.026, 0.032]	[1000, 1000]
Crude mix W	[0.048, 0.054]	[0.031, 0.039]	[0.037, 0.045]	[900, 900]
Crude mix V	[0.057, 0.061]	[0.022, 0.029]	[0.046, 0.053]	[1000, 1000]
Crude mix U	[0.064, 0.071]	[0.012, 0.019]	[0.058, 0.067]	[800, 800]
Unloading flowrate	[0, 500]	Transfer flowrate		[0, 500]
Distillation flowrate	[20, 500]	Number of distillation		16

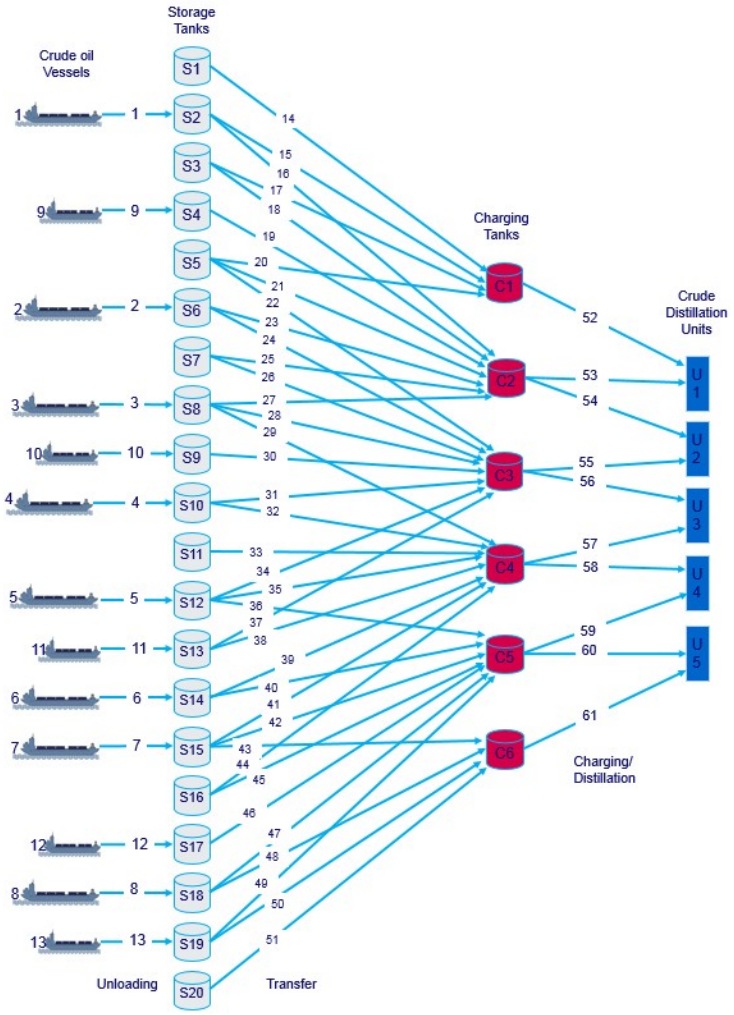

Figure 27: Crude oil operations network for COSP7

Table 27: Overview of the COSP7 data (Part 1)

Scheduling horizon				18
Vessels	Arrival time	Type	Composition	Amount of crude (Mbbl)
Vessel 1	0	VLCC	100% B	800
Vessel 2	15	VLCC	100% E	700
Vessel 3	4	VLCC	100% F	400
Vessel 4	12	VLCC	100% H	800
Vessel 5	2	VLCC	100% J	700
Vessel 6	14	VLCC	100% L	800
Vessel 7	6	VLCC	100% M	400
Vessel 8	10	VLCC	100% O	500
Vessel 9	7	Small	100% C	100
Vessel 10	3	Small	100% G	200
Vessel 11	6	Small	100% K	250
Vessel 12	16	Small	100% N	100
Vessel 13	13	Small	100% O	200
Storage tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1	[0, 900]		100% A	600
Tank 2	[100, 1100]		100% B	300
Tank 3	[0, 900]		100% B	800
Tank 4	[100, 1100]		100% C	1000
Tank 5	[0, 900]		100% D	700
Tank 6	[100, 1300]		100% E	800
Tank 7	[0, 900]		100% E	600
Tank 8	[0, 1000]		100% F	600
Tank 9	[0, 900]		100% G	500
Tank 10	[100, 1300]		100% H	700
Tank 11	[0, 900]		100% I	600
Tank 12	[0,1100]		100% J	300
Tank 13	[0,1000]		100% K	700
Tank 14	[100, 1100]		100% L	500
Tank 15	[100, 1300]		100% M	700
Tank 16	[0, 900]		100% L	800
Tank 17	[0, 900]		100% N	700
Tank 18	[0, 1000]		100% O	600
Tank 19	[0, 900]		100% O	600
Tank 20	[0, 900]		100% P	700
Charging tanks	Capacity (Mbbl)		Initial composition	Initial amount of crude (Mbbl)
Tank 1 (mix X)	[0, 1000]		100% Q	300
Tank 2 (mix Y)	[0, 1000]		100% B	100
Tank 3 (mix Z)	[0, 1000]		100% R	600
Tank 4 (mix W)	[0, 1000]		100% S	100
Tank 5 (mix V)	[0, 1000]		100% O	500
Tank 6 (mix U)	[0, 1000]		100% T	200

Table 28: Overview of the COSP7 data (Part 2)

Crudes	Property 1	Property 2	Property 3	Gross margin ($/bbl)
Crude A	0.019	0.069	0.01	1.9
Crude B	0.035	0.059	0.018	3.5
Crude C	0.038	0.054	0.023	3.8
Crude D	0.028	0.06	0.017	2.8
Crude E	0.04	0.05	0.025	4
Crude F	0.045	0.043	0.032	4.5
Crude G	0.043	0.046	0.028	4.3
Crude H	0.047	0.04	0.036	4.7
Crude I	0.051	0.035	0.041	5.1
Crude J	0.053	0.032	0.043	5.3
Crude K	0.047	0.04	0.035	4.7
Crude L	0.056	0.03	0.045	5.6
Crude M	0.059	0.024	0.048	5.9
Crude N	0.059	0.023	0.051	5.9
Crude O	0.06	0.023	0.052	6
Crude P	0.071	0.012	0.069	7.1
Crude Q	0.025	0.063	0.014	2.5
Crude R	0.043	0.044	0.029	4.3
Crude S	0.053	0.031	0.043	5.3
Crude T	0.069	0.013	0.066	6.9
Crude mixtures	Property 1	Property 2	Property 3	Demand (Mbbl)
Crude mix X	[0.020, 0.027]	[0.061, 0.069]	[0.01, 0.016]	[1500, 1500]
Crude mix Y	[0.035, 0.039]	[0.052, 0.059]	[0.018, 0.024]	[1100, 1100]
Crude mix Z	[0.041, 0.045]	[0.043, 0.049]	[0.026, 0.032]	[1500, 1500]
Crude mix W	[0.048, 0.054]	[0.031, 0.039]	[0.037, 0.045]	[1500, 1500]
Crude mix V	[0.057, 0.061]	[0.022, 0.029]	[0.046, 0.053]	[1500, 1500]
Crude mix U	[0.062, 0.071]	[0.012, 0.021]	[0.056, 0.067]	[1500, 1500]
Unloading flowrate	[0, 500]	Transfer flowrate		[0, 500]
Distillation flowrate	[20, 500]	Number of distillation		30

15 MOURET'S MIP RESULTS (15 MIN ITERATION)

Table 29: Mouret's MIP results with a time limit of 15 minutes per iteration

	Dissertation			Cardinality Rule			Improved	
Instance	n	Obj. Value	CPU(s)	n	Obj. Value	CPU(s)	n	Obj. Value	CPU(s)
COSP1	5	7975	6	5	7975	4	5	7975	2
COSP2	6	10117	753	6	10117	43	6	10117	10
COSP3	5	8740	89	5	8740	9	5	8740	5
COSP4	4	13255	245	4	13255	9	4	13255	2
COSP5	7	15656	2844	8	15658	2767	7	15656	1238
COSP6	5	27160	1801	6	27169	898	7	27168	2861
COSP7	8	42130	1806	9	42132	2709	9	42134	2532
COSP2a	6	10117	549	6	10117	38	6	10117	11
COSP5a	7	15748	2686	7	15751	1965	8	15752	2124
COSP5b	8	15751	3637	7	15751	1849	7	15751	1427
COSP6a	5	27158	1802	8	27169	3447	6	27163	1998
COSP6b	6	27164	2702	6	27169	1376	7	27168	2772
COSP7a	9	42032	2708	8	42136	1222	8	42137	1374
COSP7b	12	no solution	4509	9	42118	2706	10	42139	2790
COSP4c	4	13261	159	4	13261	5	4	13261	5
COSP5c	7	15656	2781	7	15656	1917	7	15656	1565
COSP6c	6	27168	2255	6	27168	2145	6	27168	1686
COSP7c	10	42126	3609	8	42103	1804	9	42137	2523
COSP3v	7	8740	978	7	8740	99	7	8740	20
COSP4v	6	13255	1800	6	13255	37	6	13255	10

16 CP PERFORMANCE

This chapter will present why certain choices were made on how to run the CP model.

16.1 STRENGTHENING CONSTRAINTS

One of the improvements on Mouret's MIP was removing the strengthening constraints. So we did an experiment to see if the same holds for the CP model. The results are presented in Table 30. At first glance it looks as if the results are comparable, but when we look at the objective values we see that there are two instances where a lower value was found, namely COSP3 and COSP5. This leads to the decision to also not use strengthening constraints for the CP.

Table 30: Performance CP using strengthening constraints

	Basic		Str	
Instance	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	7975	20.46	7975	20.45
COSP2	10090	44.84	10090	44.55
COSP3	8175	11.70	8166	22.61
COSP4	13247	14.25	13247	14.55
COSP5	15531	53.14	15459	54.06

16.2 SEARCH STRATEGY

The search that is used is the *Restart* search. There were two other possible choices, *Depth first* and *Multi point* but the test results, presented in Table 31, show is that the Restart search is the best choice.

Besides the search we set the search phases. Several combinations were used and several were comparable but the best choice was to set the first phase on the crude volume variables and the next phase on the start of the operation tasks. This is also a logical choice as the the level and other volume variables can be derived from crude volume, after which the precise starting point needs to be determined.

For the CP variant we add a search phase in front of the crude volume variables which is the assignment variable. The reason is that we first need to know which operation tasks there are before they can be given crude volume values and a start time.

Table 31: Comparison of the possible search strategies

	Restart		Multi Point		Depth First	
Instance	Obj. Value	CPU(s)	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	7950	21.31	7975	29.16	6025	49.18
COSP2	10114	33.96	9826	38.42	9380	59.2
COSP3	infeasible		infeasible		infeasible	
COSP4	13246	56.94	no solution		13109	10.7
COSP5	15275	58.13	no solution		no solution	
COSP6	no solution		no solution		no solution	
COSP7	no solution		no solution		no solution	
COSP2a	10112	52.25	10114	49.78	9378	12162
COSP5a	15353	59.3	no solution		no solution	
COSP5b	15383	58.5	no solution		no solution	
COSP6a	no solution		no solution		no solution	
COSP6b	no solution		no solution		no solution	
COSP7a	no solution		no solution		no solution	
COSP7b	no solution		no solution		no solution	
COSP4c	13244	36.9	no solution		13058	46.45
COSP5c	15216	52.63	no solution		no solution	
COSP6c	no solution		no solution		no solution	
COSP7c	no solution		no solution		no solution	
COSP3v	8423	48.97	8269	56.72	7970	51.77
COSP4v	13254	48.49	13172	0.12	no solution	

Table 32 presents a comparison between having no search phases and setting the search phases as was just explained. Here we see that in almost all cases the variant with search phases finds better objective values than the variant without search phases.

16.3 PRECISION

The quantities given by the data are in Mbbl (1000 barrels), meaning that if we use this quantity in CP, which has no continuous variables, we need to move crude oil (blends) in perfect quantities of Mbbl volumes. To remedy this we could increase the domain the same way we did with the time, see Section 4.6.2. This increase would allow for more possible volume movements, and thus possibly allow for better solution. So the default precision is in Mbbl but could for instance be increased to bbl (a factor 1000). However Table 33, which presents the results of a comparison between the default Mbbl precision and 100 bbl precision, shows that the increase in search space will give a far larger chance of finding a lower objective value.

Table 32: Comparison of the default setting and using search phases

	Default		Search Phases	
Instance	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	7975	10.71	7950	21.31
COSP2	10094	59.13	10114	33.96
COSP3	infeasible		infeasible	
COSP4	13196	7.3	13246	56.94
COSP5	15364	59.8	15275	58.13
COSP6	no solution		no solution	
COSP7	no solution		no solution	
COSP2a	10100	50.87	10112	52.25
COSP5a	14907	32.52	15353	59.3
COSP5b	15314	42.88	15383	58.5
COSP6a	no solution		no solution	
COSP6b	no solution		no solution	
COSP7a	no solution		no solution	
COSP7b	no solution		no solution	
COSP4c	13248	32.8	13244	36.9
COSP5c	15124	56.2	15216	52.63
COSP6c	no solution		no solution	
COSP7c	no solution		no solution	
COSP3v	8335	36.86	8423	48.97
COSP4v	13252	35.94	13254	48.49

16.4 REPLACING TOTAL VOLUME VARIABLES

The idea of only using the decision variables, by removing the derived variables from the model and insert the decision variables in their place, was already partly used to improve Mouret's MIP. This was done by removing the total volume variables and replacing it with the sum of crude volume variables. We also tested if this would help the CP.

Tables 34 presents the results of this performance test. Here we see that smaller objective values are found or that more time is needed, except for COSP2 which gives a similar result. The reason behind these results is probably because CP actually performs better with derived variables. Thus it was decided to keep the derived variables.

Table 33: Comparison of different smallest units of volume

	1 Mbbl		100 bbl	
Instance	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	7950	21.31	7961	56.85
COSP2	10114	33.96	10049	49.67
COSP3	infeasible		infeasible	
COSP4	13246	56.94	13159	57.52
COSP5	15275	58.13	no solution	
COSP6	no solution		no solution	
COSP7	no solution		no solution	
COSP2a	10112	52.25	9908	51.19
COSP5a	15353	59.3	15107	59.64
COSP5b	15383	58.5	15169	53.79
COSP6a	no solution		no solution	
COSP6b	no solution		no solution	
COSP7a	no solution		no solution	
COSP7b	no solution		no solution	
COSP4c	13244	36.9	no solution	
COSP5c	15216	52.63	no solution	
COSP6c	no solution		no solution	
COSP7c	no solution		no solution	
COSP3v	8423	48.97	no solution	
COSP4v	13254	48.49	13237	42.72

Table 34: Performance test results on the CP, total volume variable removed

	Normal		Without total volume var.	
Instance	Obj. Value	CPU(s)	Obj. Value	CPU(s)
COSP1	7975	8.17	7970	3.29
COSP2	10110	24.42	10112	26.46
COSP3	8291	15.10	8291	25.77
COSP4	13249	30.11	13213	59.37
COSP5	15530	59.03	15368	58.33

17 PROBLEM INSTANCES INPUT DATA OF THE PBDSP

This chapter presents input data of the two PBDSP instances that were used to test the optimizers.

17.1 PBDSP1

Table 35: PBDSP1: component tank data

	Component tank						
	CT1	CT2	CT3	CT4	CT5	CT6	CT7
Cost ($/bbl)	23.00	25.00	21.00	19.00	23.00	30.00	30.00
Prod. Rate(Mbbl/day)	0.96	0.768	1.776	3.84	1.00	0.00	0.00
Initial stock (Mbbl)	0.40	1.00	1.00	1.00	1.00	1.00	1.00
Min capacity (Mbbl)	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Max capacity (Mbbl)	7.00	7.00	15.00	6.209	6.209	100.00	100.00
Max outflow (Mbbl)	0.96	3.36	6.72	9.60	9.60	30.00	30.00
Component	C1	C2	C3	C4	C5	C6	C7

	CT8	CT9	CT10	CT11	CT12	CT13
Cost ($/bbl)	30.00	30.00	30.00	30.00	30.00	30.00
Prod. Rate(Mbbl/day)	0.00	0.00	0.00	0.00	0.00	0.00
Initial stock (Mbbl)	1.00	1.00	1.00	1.00	1.00	25.00
Min capacity (Mbbl)	0.00	0.00	0.00	0.00	0.00	0.00
Max capacity (Mbbl)	100.00	100.00	100.00	100.00	100.00	100.00
Max outflow (Mbbl)	30.00	30.00	30.00	30.00	30.00	30.00
Component	C8	C9	C10	C11	C12	C13

Table 36: PBDSP1: component data

	Component						
	C1	C2	C3	C4	C5	C6	C7
Property							
P1	0.57	0.71	0.69	0.77	0.78	0.66	0.68
P2	115.02	66.87	73.6	53.25	50.87	79.00	77.48
P3	97.60	98.00	98.00	89.00	88.00	68.60	68.00
P4 (NL)	63.22	2.90	13.05	1.45	2.90	10.01	11.31

	C8	C9	C10	C11	C12	C13	CB1
P1	0.80	0.87	0.66	0.66	0.73	0.75	0.74
P2	45.09	31.56	81.99	82.70	62.50	52.32	58.76
P3	97.40	100.00	68.60	81.00	71.00	110.00	95.10
P4 (NL)	6.67	1.74	10.01	11.60	10.88	7.69	8.56

Table 37: PBDSP1: product tank data

	Product Tank					
	PT1	PT2	PT3	PT4	PT5	PT6
Product	P1	P1	P3	P2	P3	P3
Initial Stock (Mbbl)	0	0	0	3.596	0	0
Min capacity (Mbbl)	0	0	0	0	0	0
Max capacity (Mbbl)	66.00	66.00	66.00	23.00	23.00	23.00
Initial Component Blend	-	-	-	CB1	-	-

Table 38: PBDSP1: product data

	Product					
	P1 (price($/bbl)=30)		P2 (price($/bbl)=31)		P3 (price($/bbl)=32)	
	Min	Max	Min	Max	Min	Max
Specifications						
P1	0.70	0.78	0.70	0.78	0.70	0.78
P2	0.00	999	0.00	999	0.00	999
P3	95.00	999	90.00	999	95.00	999
P4 (NL)	0.00	12.00	0.00	12.00	0.00	12.00

Table 39: PBDSP1: main pipeline data

	Pipeline	
	PL1	PL2
Min flowrate (Mbbl/day)	0.10	0.10
Max flowrate (Mbbl/day)	20.00	20.00

Table 40: PBDSP1: demand vessel data

	Demand Vessel							
	DV1	DV2	DV3	DV4	DV5	DV6	DV7	DV8
Product	P2	P2	P3	P3	P3	P3	P3	P1
Demand (Mbbl)	4.60	4.60	7.055	20.00	7.055	23.35	7.055	70.78
Departure	9 days	20 days	7 days	9 days	15 days	20 days	24 days	28 days

17.2 PBDSP2

Table 41: PBDSP2: component tank data

	Component Tank								
	CT1	CT2	CT3	CT4	CT5	CT6	CT7	CT8	CT9
Cost ($/bbl)	24.00	20.00	26.00	23.00	24.00	50.00	50.00	50.00	50.00
Prod. Rate (Mbbl/day)	15.00	33.00	20.00	14.00	18.00	10.00	0.00	0.00	0.00
Max outflow (Mbbl/day)	12.00	25.00	25.00	25.00	25.00	25.00	25.00	25.00	25.00
Initial stock (Mbbl)	48.00	20.00	75.00	22.00	30.00	54.00	12.00	20.00	15.00
Min capacity (Mbbl)	5.00	5.00	5.00	5.00	5.00	5.00	0.00	0.00	0.00
Max capacity (Mbbl)	100.00	250.00	250.00	100.00	100.00	100.00	100.00	100.00	100.00
Component	C1	C2	C3	C4	C5	C6	C7	C8	C9

Table 42: PBDSP2: component data

	Component								
	C1	C2	C3	C4	C5	C6	C7	C8	C9
Property									
P1 (NL)	127.27	117.83	158.97	99.48	148.19	118.61	122.14	96.75	111.08
P2 (NL)	33.81	33.34	24.67	35.13	35.76	28.99	33.57	23.52	24.71
P3	0.71	0.87	0.62	0.67	0.65	0.75	0.75	0.82	0.73
P4	3.60	1.00	100.00	94.90	91.50	15.00	0.00	1.30	34.30
P5	16.30	4.50	100.00	97.10	95.50	100.00	0.00	6.00	57.10
P6	94.30	93.50	100.00	100.00	100	100.00	0.00	93.9	95.90
P7	35.00	22.70	351.10	117.10	93.00	31.30	63.30	16.00	52.40
P8 (NL)	377.70	415.20	416.52	408.69	408.91	407.41	394.99	427.48	365.18
P9	0.00	88.60	0.00	2.30	0.20	0.00	43.98	65.30	21.30
P10	0.00	0.10	61.30	48.90	36.00	0.00	1.04	0.60	33.30
P11	0.00	3.30	0.00	1.10	0.10	0.00	3.33	0.90	0.80
P12 (NL)	10.03	11.03	8.21	12.54	12.39	12.50	12.09	7.28	7.25

Table 43: PBDSP2: product tank data

	Product Tank		
	PT1	PT2	PT3
Product	G1	G2	G3
Initial Stock (Mbbl)	0.00	0.00	0.00
Min capacity (Mbbl)	5.00	5.00	5.00
Max capacity (Mbbl)	150.00	150.00	150.00
Initial Component Blend	-	-	-

Table 44: PBDSP2: product data

	Product					
	G1 (price($/bbl)=31.00)		G2 (price($/bbl)=31.00)		G3 (price($/bbl)=31.00)	
	Min	Max	Min	Max	Min	Max
Recipe (%)						
C1	0.00	0.22	0.00	0.25	0.00	0.25
C2	0.00	0.20	0.00	0.24	0.00	0.24
C3	0.02	0.10	0.00	0.10	0.00	0.10
C4	0.00	0.06	0.00	0.23	0.00	0.23
C5	0.00	0.25	0.00	0.25	0.00	0.25
C6	0.00	0.10	0.00	0.10	0.00	0.10
C7	0.00	1.00	0.00	1.00	0.00	1.00
C8	0.00	1.00	0.00	1.00	0.00	1.00
C9	0.00	1.00	0.00	1.00	0.00	1.00
Specifications						
P1 (NL)	117.00		113.00		120.00	
P2 (NL)	31.00		30.00		31.00	
P3	0.72	0.775	0.72	0.775	0.72	0.775
P4	20.00	50.00	20.00	48.00	22.00	50.00
P5	46.00	71.00	46.00	71.00	46.00	71.00
P6	85.00		85.00		85.00	
P7	45.00	60.00	45.00	60.00	60.00	90.00
P8 (NL)		415.00		415.00		410.00
P9		42.00		42.00		42.00
P10		18.00		18.00		18.00
P11		1.00		1.00		1.00
P12 (NL)		12.50		12.00		14.00

Table 45: PBDSP2: main pipeline data

	Pipeline		
	PL1	PL2	PL3
Min flowrate (Mbbl/day)	0.00	0.00	0.00
Max flowrate (Mbbl/day)	200.00	200.00	200.00

Table 46: PBDSP2: demand vessel data

	Demand Vessel								
	DV1	DV2	DV3	DV4	DV5	DV6	DV7	DV8	DV9
Product	G1	G2	G3	G2	G1	G2	G1	G1	G3
Demand (Mbbl)	10.00	12.00	10.00	25.00	25.00	23.00	30.00	10.00	22.00
Departure	1 days	1 days	1 days	3 days	4 days	4 days	7 days	8 days	8 days

Table 47: PBDSP2: component vessel data

	Component Vessel		
	CV1	CV2	CV3
Component	C9	C8	C7
Volume (Mbbl)	50.00	85.00	90.00
Arrival	1 days	3 days	6 days