The Container Stowage Problem

Kira Janstrup (Department of Transport, Technical University of Denmark)
Trine Høyer Rose (Department of Mathematical Sciences, University of Copenhagen)
Kent Høj Andersen (Department of Mathematical Science. Århus University, Denmark)
Rune Møller Jensen (IT University of Copenhagen).

IT University Technical Report Series **TR-2010-134**

ISSN 1600–6100 **December 2010**

ISSN 1600–6100

ISBN 978-87-7949.227-1

Copies may be obtained by contacting:

IT University of Copenhagen
Rued Langgaards Vej 7
DK-2300 Copenhagen S
Denmark

Telephone: +45 72 18 50 00
Telefax: +45 72 18 50 01
Web `www.itu.dk`

The Container Stowage Problem

Kira Janstrup (Department of Transport, Technical University of Denmark)
Trine Høyer Rose (Department of Mathematical Sciences, University of Copenhagen)
Kent Høj Andersen (Department of Mathematical Science. Århus University, Denmark)
Rune Møller Jensen (IT University of Copenhagen).

1 Preface

We intended this project to be in applied mathematics and contacted therefore a larger liner shipping company. They had a container stowage problem and we found it very interesting. They do have a solution though, but they would like to know how to get better solutions, so it can be seen as an optimization problem which we will try to solve by using modelling from operations research and programming. It is therefore assumed that the reader has some knowledge of applied mathematics and operations research to better understand our project.

First a brief introduction to the project is given and why it at all is relevant to improve the solutions to the container stowage problem. In Chapter 3 based on informations from the liner shipping company we describe the different types of containers and the layout of a container vessel.

Next in Chapter 5 we simplify the container stowage problem and set extra limitations such that the first model we write up is very simple and can be solved in GAMS. In the subsequent models some of the limitations are removed so more details hence constraints will be added to the models.

The final model is implemented in the computer program CPLEX. How this and the preprocessing is done is described in Chapter 6. We present our results in Chapter 7 where we also analyze the objective function and the effect of the preprocessing. In Chapter 8 we compare our results with those from the other programming types. We finish off in Chapter 9 with a discussion of what we have achieved and what further could be interesting to do with the model and the program.

As primary sources we have used:

- Liner Shipping: Containers, Vessels, Terminals and Service Networks written by Rune M. Jensen
- Model Building In Mathematical Programming by H. P. Williams

The project is written at the Department of Mathematical Sciences at the University of Copenhagen. We want to thank our supervisor Kent Høj Andersen for guidance and give him a special thank for implementing the model in CPLEX. We also want to thank Rune Møller Jensen from the IT University in Copenhagen for giving us advice and supplying us with material. At last we want to thank Alberto Delgado-Ortegon and Dario Pacino for providing us with the data sets and sharing their results with us.

Contents

2 Introduction

The main purpose of this project is to use integer programming to create a model that minimizes the costs for container transportation by ship. To make the model as realistic as possible it will be based on information from a large shipping company about the vessel layout and container types. In addition to our project two other projects are made where an optimal solution to the container stowage problem also is tried to be found, but by using constraint programming and local search instead respectively. We will therefore in the end compare these three methods and the achieved results on fastness and optimality.

2.1 Background

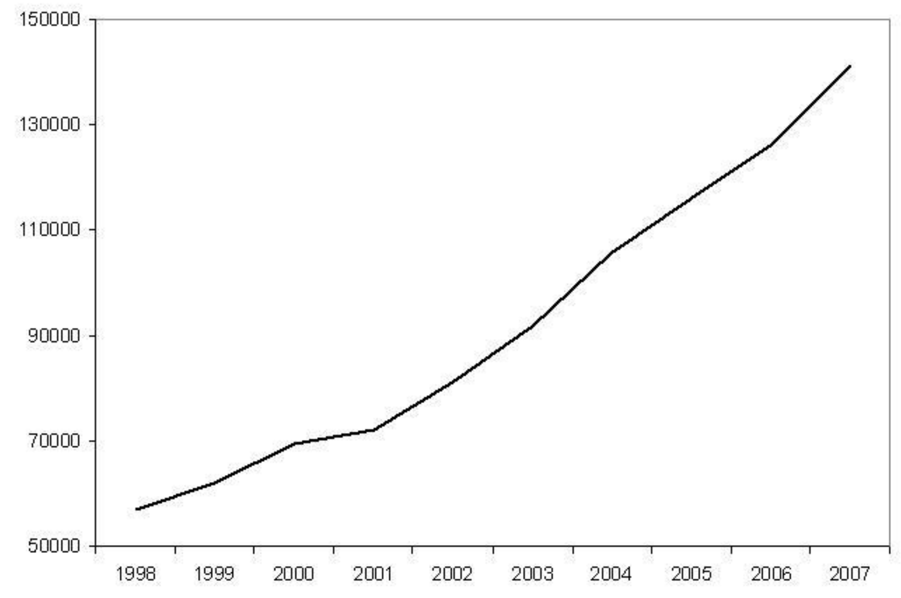


Figure 1: The global container transport by shipping in 1000 *Twenty-foot Equivalent Unit* (TEU) which are the number of 20-foot containers the ship can carry [1].

A project like this is relevant because every day on small and big destinations a huge amount of articles is transported to all over the world and by the increased globalization it is getting more and more important to do this as cheap as possible and hence as fast as possible because as the wise man says ”time is money”. The competition between the different ways for transporting goods is therefore big.

Goods transport is actually just a matter of moving some articles from one point to another but it is not always that simple and can quickly become complicated because an article is often produced of many different parts which hardly ever are made in the same place or not even in the same country. The reason for this is that the industrial companies think a lot about how they can save money and therefore find the countries where the material, labour and production respectively are cheapest. The production of one article can therefore involve a lot of transportation even before the distribution of the complete product to the final destination has started. [2]

Because of the increased trading between Asia, America and Europe goods transport by ship has become very efficient since big oceans has to be crossed. Airplanes could also be used but they can not carry as much goods as a ship, so on long distances transportation by ship is cheapest and even though it takes much longer time than with planes shipping is still to prefer. Just in the last couple of years the volume of cargo which is transported by ship has heavily increased see Figure 1.

In Denmark alone there has been an enormous increasement in the shipping industry and as seen in Table 1 the profit has almost doubled but the share of the total danish export has not increased noticeable. The shipping companies are therefore always looking for new ways to get more competitive prices and one way to do this is to to reduce the costs connected with loading containers on board ships.

Year	Billions DKK	Percent of danish export
2003	100	14
2004	120	16
2005	130	14
2006	160	16
2007	175	16

Table 1: Danish shipping companies total profit [1].

3 Container ships

Under world war two oil tankers were build in large numbers, these ships were after the end of the war useless why they were converted into container ships and the very first one could only carry 58 metal containers. Since then the development in the shipping industry has quickly increased and the variety of the ships sizes is wide where the largest ships capacity today is over 10.000 TEU. [3]

3.1 Container types

Such that all types of articles can be transported there exist many different container types and these are described below and illustrated in Figure 2.

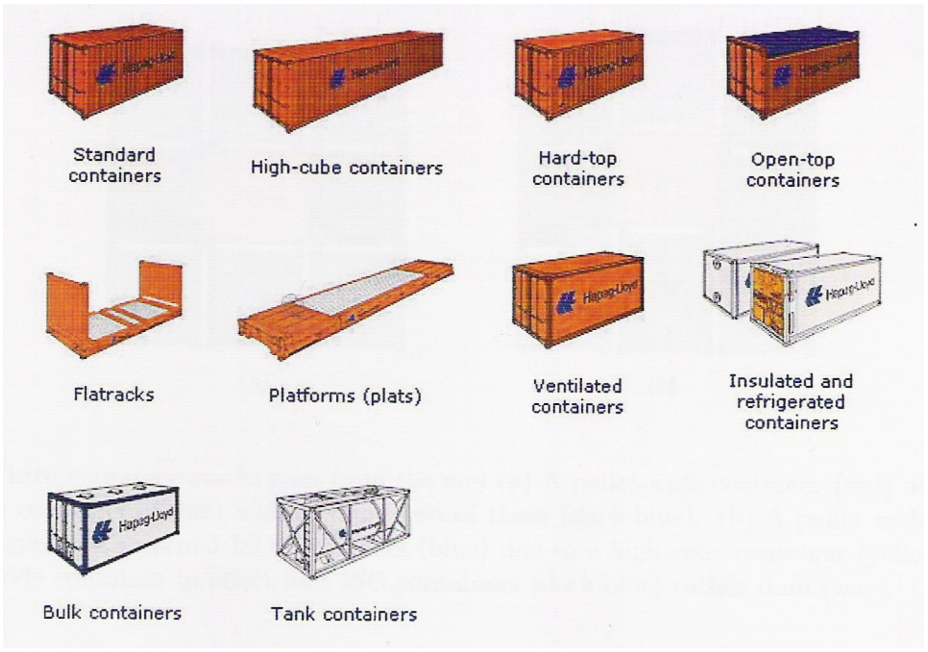


Figure 2: The different container types.

- **Standard containers.** These containers are normal steel boxes which are 8 feet wide, 8 feet 6 inches high and either 20 feet or 40 feet long.

- **Reefer/refrigerated containers.** Reefer containers are 20 feet long closed steel boxes with a built-in refrigeration system so it reduces the space for cargo. At one end of the container there is a panel to adjust the inside temperature and further there is placed a plug. The reefer container has to be placed in a cell with a power plug so the container can be connected to a power system by a cable.
- **Insulated containers.** Insulated containers do not have any integral refrigeration and can therefore hold much more cargo than a reefer container. Instead of the refrigeration system the container is insulated and has two portholes in one of the end walls. From the ship's central cooling plant cold air is blown into the container through the bottom porthole and at the top the warm air is removed hence it is possible to ensure that the cargo is staying cold.
- **High-cube containers.** A high-cube container is normally a 40-foot container which is 9 feet 6 inches high, but higher and longer containers do exist and 20-foot high-cube containers do also exist but they are very rare.
- **Hard-top containers.** The roof of a hard-top container is removable but else it is like a standard container. They are typically used for heavy and tall cargo or cargo that is easily loaded from the top.
- **Open-top containers.** Open-top containers differ from the hard-top containers in the way that they do not have any roof at all and are mostly used for tall cargo.
- **Platforms.** A steel frame with wooden floor, no side or end walls and therefore no roof is called a platform and are useful to transport very heavy and oversized cargo.
- **Flatracks.** A flatrack consists of a steel frame with soft wooden floor but different from the platform in the way that it has two end walls which can be collapsed if necessary. Flatracks exist both as 20-foot and 40-foot containers.
- **Ventilated containers.** These containers are also known as passive ventilated or coffee containers because the ventilation is provided by a small opening in the top and at the bottom.
- **Bulk containers.** A bulk container has three loading hatches in the roof and is therefore mainly used to transport gains, foodstuff and spices.
- **Tank containers.** Tank containers consist of a steel frame with a tank inside, they are mostly used to transport chemicals and foodstuffs.

Most containers are 8 feet wide but there exist containers that are a little bit wider these a called pallet-wide (PW) containers and can be hard to deal with in connection with high-cube containers and there are only a few ships which can store them under deck.

Other categories are *Out-Of-Gate* (OOG) containers where cargo sticks out, *Away-From-Heat* (AFH) containers which must be placed away from heated fuel tanks, *Requiring-Power-Plug* (RPP) containers that are non-reefers but still require a power plug and IMO containers that contains hazardous cargo. They are called so because it is the *International Maritime Organization* that sets the rules for how they must be stored. IMOs are numbered one to four which describes how dangerous the cargo is and how it must be handled. For IMO containers in risk group 1 there are no specific rules, IMO containers in risk group 2, 3 and 4 must be separated by 1, 2 and 3 non-IMO containers respectively.[4][1]

4 The container vessel layout

It is important to know where exactly a container on the ship is placed since we want to know where to find it when it has to be offloaded, therefore the container vessel is split into several areas as shown in figure 3. The cargo space is split into a *under deck* and *over deck* by a number of *hatch covers* to prevent the area under deck to be filled with water.

[1]The reference applies to the whole section.

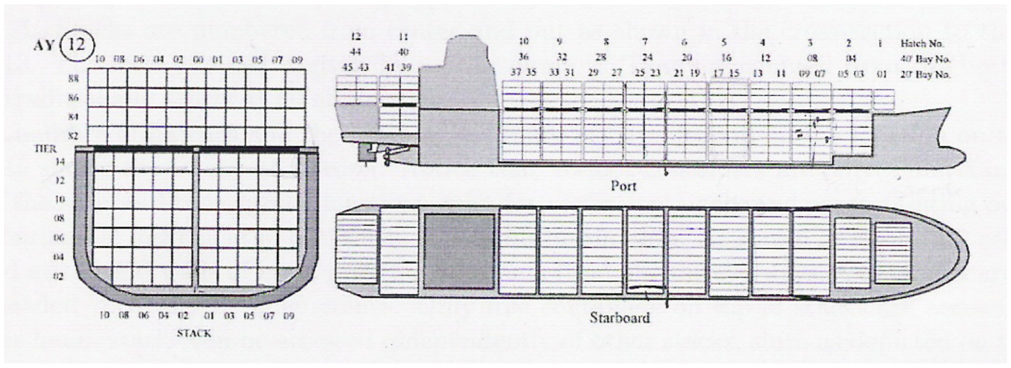


Figure 3: The stowage arrangement of the ship. (Figure 13 from [4])

A *bay* is a subsection across the ship consisting of *stacks* which are 40 feet long and 8 feet wide but as the bay outline in Figure 4 illustrates their height varies with their position in the bay. A subcollection of stacks in one bay is called a *location*. The stack is divided into a number of *cells* of height 8 feet 6 inches so the capacity of a cell is two TEU or one Forty-foot Equivalent Unit (FEU). Bays are numbered from stern to bow where the bays that only can hold 20-foot containers are given odd numbers. Stacks are numbered from center and out and the cells are given by it's row number see Figure 3.

In Figure 4 some of the cells are marked with a triangle if there can be placed a reefer container. These cells are normally in the bottom of a stack under deck and have one or two power plug for the reefer containers. When placing

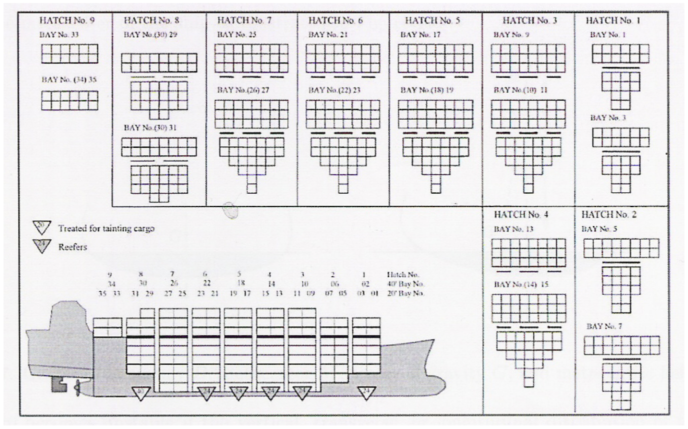


Figure 4: An example of a bay layout. (Figure 15 from [4])

the containers at the ship it is important to maintain the weight balance therefore the distribution of cargo is normally done as illustrated in Figure 5, so heavy cargo is placed at the center, as even distribution on the stern and the bow or as uneven transverse distribution. [4][2]

5 Models of the stowage problem

The motivation in this chapter is to present a mathematical model which gives an optimal solution regarding container stowage on board a container vessel. It would be very difficult if at all possible to model the whole ship, and if we succeeded we would end up with a big problem that would be impossible to solve, we therefore only look at a section

[2]The reference applies to the whole section.

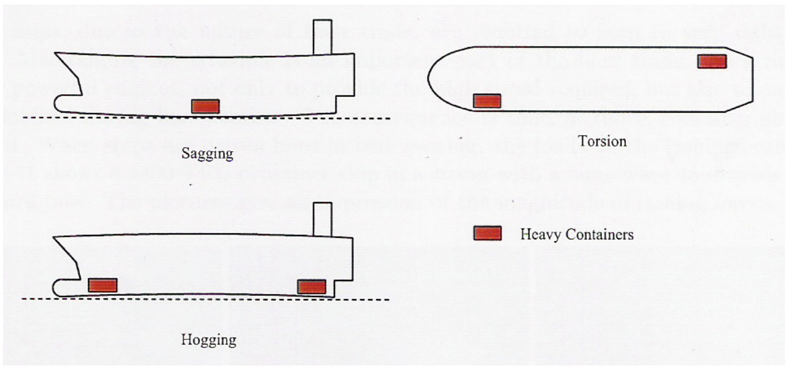


Figure 5: Distribution of heavy cargo. (Figure 19 from [4])

Constant	Denotes	Values
J	Number of stacks	$\mathbb{Z}_+$
D	Number of destinations	$\mathbb{Z}_+$
L_{20}	Number of $20'$ containers to be loaded	$\mathbb{Z}_+$
L_{40}	Number of $40'$ containers to be loaded	$\mathbb{Z}_+$

Table 2: The applied constants in the model.

of a bay. Further we only consider under deck and since several of the containers are only suitable to be placed over deck we do not include all the different types.

We start with building a very simple model with only a few constraints and then using the program GAMS to check the validity off the model. When ensured the model is working as intended, we will add more constraints and by continuing in this way we will at last end up with a model including enough details such that a solution can be used to the original real world problem. The main purpose of the model is to minimize the cost in general and with regard to this one of the most essential problems is to avoid moving already stored containers on the ship. This can be achieved by in general avoiding overstowing which is when a container is in the way of another meaning that it is necessary to move it to get a container underneath with an earlier destination port off. Therefore we also want to keep as many empty stacks as possible such that when loading new containers at later ports they will not be in the way of containers already stored at the ship. Further it is desirable that all the containers in a stack have the same destination port and these stacks are called pure stacks. The last thing to do for helping avoiding unnecessary moves is to minimize the number of non-reefer containers in reefer cells, so the risk of later standing without a reefer cell for a reefer container is minimized.

5.1 Model 1

In the first model we only include 20- and 40-foot containers and assume that the maximal difference between fore and aft in a stack can only be one container. We will also just consider one loading port and pay no attention to the destination ports of the containers. To build the model we first need the constants in Table 2 to define the size of the section of the ship, number of destinations and the number of containers to be loaded. Further we in Table 3 introduce a parameter which denotes the number of cells in stack J.

In Table 4 we apply the variables that indicate the number of 20- and 40-foot containers respectively in a cell and

Parameter	Denotes	Values
K_j	Number of cells in stack J	$\mathbb{Z}_+$

Table 3: The applied parameter in the model.

Variable	Denotes	Values
t_{jk}	Number of $20'$ containers in cell k in stack j	$0, 1,$ and 2
f_{jk}	Number of $40'$ containers in cell k in stack j	0 and 1
I	The total number of containers at the ship	$\sum_{j=1}^{J}\sum_{k=1}^{K_j}(t_{jk}+f_{jk})$

Table 4: The applied variables with given values.

the total number of containers at the ship.

Since we here have a very simple model and only consider one destination port we cannot talk about overstowing, hence it is very difficult to give a meaningful objective function. We will therefore let the objective function be $\theta = I$ and thereby maximize the number of filled cells at the ship.

When loading the ship the following conditions has to be satisfied

1. All $20'$ containers that has to be loaded are loaded.
2. All $40'$ containers that has to be loaded are loaded.
3. To be able to place a $40'$ container in a cell the cell below has to be full.
4. To be able to place one or two $20'$ containers in a cell there has to be two $20'$ containers in the cell below. This also ensures that a $20'$ container will not be placed on the top of a $40'$ container.
5. If there are placed a $40'$ container in a cell there cannot be placed any $20'$ containers in the concerned cell.
6. If there are placed one or two $20'$ containers in a cell there cannot be placed any $40'$ container in the concerned cell.

Written in mathematical therms the conditions are as follows

$$\begin{aligned}
\sum_{j=1}^{J}\sum_{k=1}^{K_j} t_{jk} &= L_{20} && (1)\\
\sum_{j=1}^{J}\sum_{k=1}^{K_j} f_{jk} &= L_{40} && (2)\\
f_{jk} = 1 \quad &\Rightarrow \quad f_{j(k-1)} = 1 \vee t_{j(k-1)} = 2 && (3)\\
t_{jk} \geq 1 \quad &\Rightarrow \quad t_{j(k-1)} = 2 && (4)\\
f_{jk} = 1 \quad &\Rightarrow \quad t_{jk} = 0 && (5)\\
t_{jk} \geq 1 \quad &\Rightarrow \quad f_{jk} = 0. && (6)
\end{aligned}$$

and they should be valid for all $j = 1, \ldots, J$ and for all $k = 1, \ldots, K_j$. The equations (3) to (6) cannot be programmed in GAMS since they are not linear, so it is necessary to paraphrase these equations.

For paraphrasing (3) we use from Williams [6] that the linear form of

$$\rho = 1 \quad \Rightarrow \quad \rho_1 = 1 \vee \rho_2 = 1$$

is

$$\rho_1 + \rho_2 - \rho \quad \geq \quad 0, \qquad (7)$$

where all the variables are binary. Since t_{jk} takes the values 0, 1, and 2 it is not a binary variable, but with a small adaption (7) can still be used and condition (3) is therefore written linearly as

$$t_{j(k-1)} + 2f_{j(k-1)} - 2f_{jk} \quad \geq \quad 0 \qquad \forall j \, \forall k.$$

To rewrite the other equations we define the binary variable

$$\delta_{jk} = \begin{cases} 0 & , \quad t_{jk} = 0 \\ 1 & , \quad t_{jk} \geq 1. \end{cases}$$

To express this variable by linear equations we use from [6] that an equation of the form

$$a \geq b \quad \Leftrightarrow \quad \rho = 1$$

is written linearly as

$$\begin{aligned} a + m\rho \quad &\geq \quad m + b \\ a - (M + \varepsilon)\rho \quad &\leq \quad b - \varepsilon, \end{aligned} \tag{8}$$

where ρ is a binary variable, b is a constant, a is a continuous variable, and ε is greater than zero. m and M are lower and upper bound of $a - b$ respectively, so in our case where $a = t_{jk}$ and $b = 1$ we get $m = -1$ and $M = 1$. Since we consider integer variables ε is set equal to 1.

We include δ_{jk} in the model by the equations

$$\begin{aligned} t_{jk} - \delta_{jk} \quad &\geq \quad 0 \qquad \forall j \, \forall k \\ t_{jk} - 2\delta_{jk} \quad &\leq \quad 0 \qquad \forall j \, \forall k. \end{aligned}$$

With this variable equation (4) to (6) for all $j = 1, \ldots, J$ and for all $k = 1, \ldots, K_j$ becomes

$$\delta_{jk} = 1 \quad \Rightarrow \quad t_{j(k-1)} = 2 \tag{9}$$

$$f_{jk} = 1 \quad \Rightarrow \quad \delta_{jk} = 0 \tag{10}$$

$$\delta_{jk} = 1 \quad \Rightarrow \quad f_{jk} = 0, \tag{11}$$

which also makes it easier to rewrite these equations to linear form. Williams [6] gives us that with binary variables the linear equation of

$$\rho_1 = 1 \quad \Rightarrow \quad \rho_2 = 1$$

is

$$\rho_1 - \rho_2 \quad \leq \quad 0. \tag{12}$$

In (9) $t_{j(k-1)} = 2$ but with a small change the above paraphrasing can still be used and (9) is then expressed linearly as

$$2\delta_{jk} - t_{j(k-1)} \quad \leq \quad 0 \qquad \forall j \, \forall k.$$

Equations (10) and (11) can now be represented by one equation because they are equivalent to that either $f_{jk} = 1$ or $\delta_{jk} = 1$. Linearly it is written as

$$f_{jk} + \delta_{jk} \quad \leq \quad 1 \qquad \forall j \, \forall k. \tag{13}$$

With the two equations for δ_{jk} and the paraphrasing of the equations, our first linear model becomes

$$\begin{array}{llcll} \max & \theta = I & & & \\ \text{s.t.} & t_{jk} - \delta_{jk} & \geq & 0 & \forall j \, \forall k \\ & t_{jk} - 2\delta_{jk} & \leq & 0 & \forall j \, \forall k \\ & \sum_{j=1}^{J} \sum_{k=1}^{K_j} t_{jk} & = & L_{20} & \\ & \sum_{j=1}^{J} \sum_{k=1}^{K_j} f_{jk} & = & L_{40} & \\ & t_{j(k-1)} + 2f_{j(k-1)} - 2f_{jk} & \geq & 0 & \forall j \, \forall k \\ & 2\delta_{jk} - t_{j(k-1)} & \leq & 0 & \forall j \, \forall k \\ & f_{jk} + \delta_{jk} & \leq & 1 & \forall j \, \forall k. \end{array}$$

Parameter	Denotes	Values
s_i	Whether container i is $20'$ long	1, it is 0, it is not
l_i	Whether container i is $40'$ long	1, it is 0, it is not
h_i	The height in foot of container i	8.5 and 9.5
w_i	The weight of container i in kilogram	$\mathbb{Z}_+$
r_i	If container i is a reefer container	1, reefer 0, not a reefer

Table 5: Parameters for each container.

Variable	Denotes	Values
c_{jki}	Whether container i is placed in cell k in stack j	1, it is there 0, it is not there
ch_{jk}	The height in foot of the container in cell k in stack j	8.5 and 9.5

Table 6: Variables for each cell.

5.1.1 Programming in GAMS

We want to check that the linear model works as intended and for this we use GAMS. First we need to define the sets for the number of stacks and cells. Since the set for the number of cells varies with the stack number we would like it to take values in $\{1, \ldots, K_j\}$ where K_j is a given parameter. This is not possible in GAMS because it needs a specific number for a set's definition area, hence for each stack we need to define a set for the number of cells. We therefore choose only to model a small section of two stacks with the set $K1$ and $K2$ with two cells and three cells respectively, see Appendix B. Since we now have different sets for each stack and a variable can only be assigned to one set in GAMS, we need to define a variable and the belonging conditions for both $K1$ and $K2$ for each variable that depend on the number of cells in a stack. When programming the condition ensuring that before placing a container in cell k the cell $k-1$ has to be full, we run into further problems with GAMS because as mentioned when a variable for example depend on the set $K1$ with elements k it can not also depend on the set $K1-1$ with elements $k-1$. The way we got around this was to use the command `alias` to make a similar set $m1$ to $K1$ and then define the equation only to be true when the order of an element in $m1$ is smaller than the order of an element in $K1$. In the theory this changes the condition to that all cells underneath cell k has to be full but indirectly the original condition also demands this, hence it does not change the model. For full details of the program see Appendix B and for the result see Appendix C. The program works as intended and gives an optimal solution for this small model, but for every stack added we need a new set of variables and conditions so very fast the typing will be tedious and the program will become very big. Especially when we extend the model and include more variables and conditions, GAMS will be a very inefficient tool for solving the model and maybe it will not even be possible to type up our final model in GAMS. We will therefore instead use CPLEX for the further programming.

5.2 Model 2

In the first extension of the model we include height and weight of the containers and whether they are reefer containers or not. We will though only consider standard containers at $8'$ $6''$ and high cube containers at $9'$ $6''$. There do exist $20'$ high cube containers but they are quite rare so we will ignore them. Therefore $20'$ containers will always be $8'$ $6''$ high in our model hence there is no need for a constraint that guaranties that two neighbouring $20'$ containers in a cell have the same hight. To express the other constraints that are necessary with these extensions we introduce the parameters in Table 5 denoting the dimensions, weight and type of the containers. In Table 6 are listed the variables indicating which container is placed in the cell and in Table 7 the parameters for the cell type and the limits of the stack.

With the new extensions the following constraints are needed to ensure that each container is placed correctly.

Parameter	Denotes	Values
r_{jk}	Number of plugs for a reefer container in cell k in stack j	0, 1 and 2
W_j	The maximum allowed weight in kilogram of the containers in stack j	$\mathbb{Z}_+$
H_j	The under deck height at the ship of stack j in foot	$8.5 \cdot K_j$

Table 7: Parameters for each stack and cell.

1. A container is placed in one and only one cell

$$\sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{jki} \quad = \quad 1 \qquad \forall i.$$

2. A cell can maximum hold two containers

$$\sum_{i=1}^{I} c_{jki} \quad \leq \quad 2 \qquad \forall j \, \forall k.$$

3. The number of $20'$ containers in cell k in stack j becomes

$$\sum_{i=1}^{I} c_{jki} s_i \quad = \quad t_{jk} \qquad \forall j \, \forall k.$$

4. The number of $40'$ containers in cell k in stack j becomes

$$\sum_{i=1}^{I} c_{jki} l_i \quad = \quad f_{jk} \qquad \forall j \, \forall k.$$

5. The number of reefer containers in a cell may not exceed the number of plugs in the cell

$$\sum_{i=1}^{I} c_{jki} r_i \quad \leq \quad r_{jk} \qquad \forall j \, \forall k.$$

6. The total weight of the containers in stack j may not exceed the maximum allowed weight of stack j

$$\sum_{k=1}^{K_j}\sum_{i=1}^{I} c_{jki} w_i \quad \leq \quad W_j \qquad \forall j.$$

7. The container height of the cell becomes the height of container i.

$$c_{jki} = 1 \quad \Rightarrow \quad ch_{jk} = h_i \qquad \forall j \, \forall k \, \forall i.$$

8. The total height of the containers in stack j may not exceed the maximum allowed height og stack j.

$$\sum_{k=1}^{K_j} ch_{jk} \quad \leq \quad H_j \qquad \forall j.$$

The condition where we define the container height of the cell is not linear why we split it into two

$$\begin{aligned} c_{jki} = 1 \quad &\Rightarrow \quad ch_{jk} \geq h_i \qquad \forall j \, \forall k \, \forall i \\ c_{jki} = 1 \quad &\Rightarrow \quad ch_{jk} \leq h_i \qquad \forall j \, \forall k \, \forall i. \end{aligned}$$

By using (8) first with $a = ch_{jk}$ and $b = h_i$, and afterwards with $a = h_i$ and $b = ch_{jk}$ these equations is paraphrased to the linear equations

$$\begin{aligned} ch_{jk} - 9.5c_{jki} - h_i \quad &\geq \quad -9.5 \qquad \forall j \, \forall k \, \forall i \\ h_i - ch_{jk} - c_{jki} \quad &\geq \quad -1 \qquad \forall j \, \forall k \, \forall i. \end{aligned}$$

In our model we have allowed a non-reefer container to be placed in a reefer cell therefore we set the objective function to minimize the number of times this happens so the risk of later standing without room in a reefer cell for a reefer container is minimized. The equation

$$\sum_{j=1}^{J} \sum_{k=1}^{K_j} \big(r_{jk} - \sum_{i=1}^{I} r_i c_{jki} \big)$$

gives the number of unused plugs in cell k in stack j, but if the cell is empty r_{jk} is still added in the objective function as if there were placed non-reefers container in the cell. To avoid this we multiply r_{jk} by $(f_{jk} + \delta_{jk})$ such that the objective function becomes

$$\sum_{j=1}^{J} \sum_{k=1}^{K_j} \big((f_{jk} + \delta_{jk}) r_{jk} - \sum_{i=1}^{I} r_i c_{jki} \big).$$

Since l_i and s_i define whether a container is $20'$ or $40'$ long and we have ensured that all containers are placed in a cell we no longer need the equations from model 1 ensuring that the containers which are supposed to be loaded has been loaded. Hence the full second model including the rest of the constraints from model 1 becomes

$$\begin{array}{llcll} \min & \theta = \sum_{j=1}^{J} \sum_{k=1}^{K_j} \big((f_{jk} + \delta_{jk}) r_{jk} - \sum_{i=1}^{I} r_i c_{jki} \big) & & & \\ \text{s.t.} & t_{jk} - \delta_{jk} & \geq & 0 & \forall j \, \forall k \\ & t_{jk} - 2\delta_{jk} & \leq & 0 & \forall j \, \forall k \\ & t_{j(k-1)} + 2f_{j(k-1)} - 2f_{jk} & \geq & 0 & \forall j \, \forall k \\ & 2\delta_{jk} - t_{j(k-1)} & \leq & 0 & \forall j \, \forall k \\ & f_{jk} + \delta_{jk} & \leq & 1 & \forall j \, \forall k \\ & \sum_{j=1}^{J} \sum_{k=1}^{K_j} c_{jki} & = & 1 & \forall i \\ & \sum_{i=1}^{I} c_{jki} & \leq & 2 & \forall j \, \forall k \\ & \sum_{i=1}^{I} c_{jki} s_i & = & t_{jk} & \forall j \, \forall k \\ & \sum_{i=1}^{I} c_{jki} l_i & = & f_{jk} & \forall j \, \forall k \\ & \sum_{i=1}^{I} c_{jki} r_i & \leq & r_{jk} & \forall j \, \forall k \\ & \sum_{k=1}^{K_j} \sum_{i=1}^{I} c_{jki} w_i & \leq & W_j & \forall j \\ & ch_{jk} - 9.5c_{jki} - h_i & \geq & -9.5 & \forall j \, \forall k \, \forall i \\ & h_i - ch_{jk} - c_{jki} & \geq & -1 & \forall j \, \forall k \, \forall i \\ & \sum_{k=1}^{K_j} ch_{jk} & \leq & H_j & \forall j. \end{array}$$

5.3 Model 3

To be able to compare our results with those achieved from the other programming methods, we all need to have the same restrictions. Therefore we now simplify our model such that a cell cannot only hold one $20'$ container, i.e. there

Parameter	Denotes	Values
a_{id}	Whether container i has destination d	1, it has 0, it has not

Table 8: Parameter concerning destination ports.

Variable	Denotes	Values
O_{jk}	Whether the container in cell k in stack j is causing overstowing	1, it is 0, it is not
δ_{jkd}	Whether there is a container below cell k in stack j with a destination before d	1, if $\sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'}c_{jk'i} \geq 1$ 0, else

Table 9: Variables used for defining overstowing.

will either be none or two. So we will now only let t_{jk} take the values 0 and 1, where 1 means there are two containers. This is implemented in the model by substituting t_{jk} by $2t_{jk}$ and δ_{jk} by t_{jk} in the constraints. When we know there will never be only one $20'$ container in a cell the height limit of a stack can now be written as

$$\sum_{k=1}^{K_j}\sum_{i=1}^{I}(\tfrac{1}{2}(c_{jki}h_is_i) + c_{jki}h_il_i) \quad \leq \quad H_j \qquad \forall j.$$

In our previous model our objective function minimized the number of non-reefer containers in reefer cells, but as mentioned earlier this is only one of the main issues in the stowage problem. We would therefore like to include terms concerning overstowing in our next objective function.

To be able to talk about overstowing, we need to include more ports in our model. For each container the parameter a_{id} indicates if the container is getting off at port d (Table 8), but we still assume that all containers are loaded at port one. Furthermore a binary variable, O_{jk}, that indicates whether the container in the cell is causing overstowing is introduced. This is the case if there is one or more containers in the cells below with earlier destination ports. We therefore define a variable, δ_{jkd}, which takes the value one if there underneath cell k in stack j is at least one container that is getting off before port d. These variables are listed in Table 9.

We know that a binary variable can be expressed by a linear equation of the form (8) so δ_{jkd} is included in the model by the equation

$$\sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'}c_{jk'i} - 2(k-1)\delta_{jkd} \quad \leq \quad 0 \qquad \forall j\,\forall k\,\forall d.$$

We express O_{jk} by the following equation

$$a_{id}c_{jki} = 1 \wedge \delta_{jkd} = 1 \quad \Rightarrow \quad O_{jk} = 1 \qquad \forall j\,\forall k\,\forall d \tag{14}$$

which we from Williams [6] know has the linear form

$$a_{id}c_{jki} + \delta_{jkd} - O_{jk} \quad \leq \quad 1 \qquad \forall j\,\forall k\,\forall d\,\forall i.$$

In the objective function we want to minimize the sum of O_{jk}, so whenever it is possible it will be set to zero why we do not need both implications in (14). We do not either need two equations for δ_{jkd} because if it is not forced to be one it will be set to zero such that O_{jk} can be zero.

For defining empty and pure stacks we use the two variables in Table 10. By these definitions the sum of E_j is the number of stacks that has been used, and the sum over d of P_{jd} is the number of different destinations that the containers in stack j have, so it is minimum one which imply that the stack is pure. They are added to the model by the equations

$$\begin{aligned} E_j - c_{jki} \quad &\geq \quad 0 \qquad \forall j\,\forall k\,\forall i \\ P_{jd} - a_{id}c_{jki} \quad &\geq \quad 0 \qquad \forall j\,\forall k\,\forall i\,\forall d \end{aligned}$$

Variable	Denotes	Values
E_j	Whether stack j is empty	1, it is not 0, it is
P_{jd}	Whether there is a container with destination d in stack j	1, there is 0, there is not

Table 10: Variables for empty stacks and pure stacks.

Objective term	Penalty
$\sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk}$	1000
$\sum_{j=1}^{J}\sum_{d=2}^{D} P_{jd}$	200
$\sum_{j=1}^{J} E_j$	100
$\sum_{j=1}^{J}\sum_{k=1}^{K_j}\big((t_{jk}+f_{jk})r_{jk}-\sum_{i=1}^{I} r_i c_{jki}\big)$	50

Table 11: The penalties for the objectives.

where we have used (12) for the paraphrasing. It is also desirable to minimize the sum of each of these variables in the objective function, so they too will be set to zero whenever it is possible hence only one equation for each is needed.

One way to include the above three objectives and the one used in model 2 in one objective function is to give each term a penalty in regard to in which order we prioritize to minimize them. Overstowing is the most important issue to minimize therefore it is given the highest penalty. This and the rest of the penalties are given in Table 5.3.

The final model then becomes

$$\begin{aligned}
\min \quad & \theta = 1000\sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk} + 200\sum_{j=1}^{J}\sum_{d=2}^{D} P_{jd} \\
& + 100\sum_{j=1}^{J} E_j + 50\sum_{j=1}^{J}\sum_{k=1}^{K_j}\big((t_{jk}+f_{jk})r_{jk} - \sum_{i=1}^{I} r_i c_{jki}\big)
\end{aligned}$$

$$\begin{array}{lrcll}
\text{s.t.} & t_{j(k-1)} + f_{j(k-1)} - f_{jk} & \geq & 0 & \forall j\,\forall k \\
& t_{jk} - t_{j(k-1)} & \leq & 0 & \forall j\,\forall k \\
& t_{jk} + f_{jk} & \leq & 1 & \forall j\,\forall k \\
& \sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{jki} & = & 1 & \forall i \\
& \sum_{i=1}^{I} c_{jki} & \leq & 2 & \forall j\,\forall k \\
& \sum_{i=1}^{I} c_{jki}s_i & = & 2t_{jk} & \forall j\,\forall k \\
& \sum_{i=1}^{I} c_{jki}l_i & = & f_{jk} & \forall j\,\forall k \\
& \sum_{i=1}^{I} c_{jki}r_i & \leq & r_{jk} & \forall j\,\forall k \\
& \sum_{k=1}^{K_j}\sum_{i=1}^{I} c_{jki}w_i & \leq & W_j & \forall j \\
& \sum_{k=1}^{K_j}\sum_{i=1}^{I}(\frac{1}{2}(c_{jki}h_is_i) + c_{jki}h_il_i) & \leq & H_j & \forall j \\
& \sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'}c_{jk'i} - 2(k-1)\delta_{jkd} & \leq & 0 & \forall j\,\forall k\,\forall d \\
& a_{id}c_{jki} + \delta_{jkd} - O_{jk} & \leq & 1 & \forall j\,\forall k\,\forall d\,\forall i \\
& E_j - c_{jki} & \geq & 0 & \forall j\,\forall k\,\forall i \\
& P_{jd} - a_{id}c_{jki} & \geq & 0 & \forall j\,\forall k\,\forall i\,\forall d
\end{array}$$

Instead of including all four objectives with the given penalties at once, we can solve the problem with only one objective at the time and after each solution add another condition and then solve the new model with the next objective in the order they are prioritized. So we start with minimizing overstowing hence the objective function is

Set	Definition	Number of elements
S	The set of containers at the ship	$\sum_{j=1}^{J}\sum_{k=1}^{K_j}(2t_{jk}+f_{jk})$
T	The set of $20'$ containers at the ship	$2\sum_{j=1}^{J}\sum_{k=1}^{K_j}t_{jk}$
F	The set of $40'$ containers at the ship	$\sum_{j=1}^{J}\sum_{k=1}^{K_j}f_{jk}$

Table 12: The sets of containers.

$\theta_1 = \sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk}$. If the optimal solution is found to be $\theta_1 = x$, we either add the condition

$$\sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk} \quad \leq \quad x$$

to the model or fix the O_{jk} variables to the found optimal values. Continuing in this way for all the objectives we will end up with the same result as if all were included at once.

5.3.1 Alternative model

When programming the model it can be an advantage to have fewer variables because it might make the program faster. In model 2 we introduced the variable c_{jki} and used it to express t_{jk} and f_{jk} respectively, so using those expression we can substitute t_{jk} and f_{jk} out of the model. Further if we divide the set of containers at the ship into two subsets T and F as defined in Table 5.3.1, we do not either need s_i and l_i. Using these changes the expressions for t_{jk} and f_{jk} from model 3 then becomes

$$\tfrac{1}{2}\sum_{i\in T} c_{jki} \quad = \quad t_{jk} \qquad \forall j\ \forall k \tag{15}$$

$$\sum_{i\in F} c_{jki} \quad = \quad f_{jk} \qquad \forall j\ \forall k. \tag{16}$$

In model 3 t_{jk} and f_{jk} are defined only to take the values 0 and 1, so if we use the above expressions instead of the variables we need constraints setting these limits. T is a subset of S, and we have a constraint saying that the limit of containers in a cell is two, hence the left hand side in (15) will also have two as a limit. By adding

$$\sum_{i'\in T} c_{jki'} \quad \geq \quad 2c_{jki} \qquad \forall j\ \forall k\ \forall i$$

(15) will only take the right values. For the $40'$ containers we just need to add that the sum of c_{jki} over F is maximum one. Now the model looks like

$$\begin{array}{llll}
\min & \multicolumn{3}{l}{\theta = 1000\sum_{j=1}^{J}\sum_{k=1}^{K_j} O_{jk} + 200\sum_{j=1}^{J}\sum_{d=2}^{D} P_{jd} + 100\sum_{j=1}^{J} E_j} \\
& \multicolumn{3}{l}{+\ 50\sum_{j=1}^{J}\sum_{k=1}^{K_j}\big((\tfrac{1}{2}\sum_{i\in T} c_{jki} + \sum_{i\in F} c_{jki})r_{jk} - \sum_{i=1}^{I} r_i c_{jki}\big)} \\
\text{s.t.} & \frac{1}{2}\sum_{i\in T} c_{j(k-1)i} + \sum_{i\in F} c_{j(k-1)i} - \sum_{i\in F} c_{jki} & \geq 0 & \forall j\,\forall k \\
& \frac{1}{2}\sum_{i\in T} c_{jki} - \frac{1}{2}\sum_{i\in T} c_{j(k-1)i} & \leq 0 & \forall j\,\forall k \\
& \frac{1}{2}\sum_{i\in T} c_{jki} + \sum_{i\in F} c_{jki} & \leq 1 & \forall j\,\forall k \\
& \sum_{j=1}^{J}\sum_{k=1}^{K_j} c_{jki} & = 1 & \forall i \\
& \sum_{i=1}^{I} c_{jki} & \leq 2 & \forall j\,\forall k \\
& \sum_{i'\in T} c_{jki'} & \geq 2c_{jki} & \forall j\,\forall k\,\forall i \\
& \sum_{i\in F} c_{jki} & \leq 1 & \forall j\,\forall k \\
& \sum_{i=1}^{I} c_{jki} r_i & \leq r_{jk} & \forall j\,\forall k \\
& \sum_{k=1}^{K_j}\sum_{i=1}^{I} c_{jki} w_i & \leq W_j & \forall j \\
& \sum_{k=1}^{K_j}\sum_{i=1}^{I}(\frac{1}{2}(c_{jki}h_i s_i) + c_{jki}h_i l_i) & \leq H_j & \forall j \\
& \sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'}c_{jk'i} - 2(k-1)\delta_{jkd} & \leq 0 & \forall j\,\forall k\,\forall d \\
& a_{id}c_{jki} + \delta_{jkd} - O_{jk} & \leq 1 & \forall j\,\forall k\,\forall d\,\forall i \\
& E_j - c_{jki} & \geq 0 & \forall j\,\forall k\,\forall i \\
& P_{jd} - a_{id}c_{jki} & \geq 0 & \forall j\,\forall k\,\forall i\,\forall d
\end{array}$$

6 Implementation of the model

As we have mentioned earlier we will only run the program for one location in a bay at the time. The provided data sets, see Appendix D, include information about the size of the location, weight and height limits for the stacks, the type of the cells, and where in the location containers already has been placed. Further it contains all the details about the containers that are loaded and those that has to be loaded. CPLEX can not read this format so a header file, see Appendix E, is created that defines the parameters and constants from the data file. It also defines functions to read the data and others to print the outline of our model. This file is included in a source code that subtracts the data and specifies the functions defined in the header file. It is also here our model is programmed and the layout of the output files are defined. Our equation for the term in the objective for non-reefer containers in reefer cells was changed in the program such that it was limited to work for reefer cells with only one plug. It is also all right here because none of the data sets has more than one reefer plug, but we want it to work in general so we have programmed our original equation, see Appendix F. The main file that converts the problem into an LP format that CPLEX can read and solve is in Appendix G. In this file it is specified which data set is the input file, and the actual CPLEX commands for the solution processes are also given here. Further explanation of the codes can be found in each of the appendices.[3]

6.1 Preprocessing

In CPLEX the branch and bound method is used to solve the problem so it starts by solving the LP-relaxation. CPLEX does run a presolver first, but here we can also manually create cuts. The equation we use to define the variable δ_{jkd} can be changed and by moving one of the terms the existing equation looks like

$$\sum_{k'=1}^{k-1}\sum_{d'=2}^{d-1}\sum_{i=1}^{I} a_{id'}c_{jk'i} \quad \leq \quad 2(k-1)\delta_{jkd} \qquad \forall j\,\forall k\,\forall d. \tag{17}$$

It will be a stronger equation if it has to be true for each term such that it becomes

$$a_{id'}c_{jk'i} \quad \leq \quad \delta_{jkd} \qquad \forall j\,\forall k\,\forall d\,\forall i\,\forall k'\,\forall d' \tag{18}$$

[3]Kent Høj Andersen made the full program but we have made some changes we found necessary.

where $k' \in \{1, \dots, k-1\}$ and $d' \in \{2, \dots, d-1\}$. The two constraints are the same when the variables have to be integers but when they are allowed to be real numbers, (17) can be true even though some variables do not satisfy (18). So these solutions are ruled out but no integer solutions are cut away because a solution where all variables satisfy (18) will also be true for (17).

In (18) we multiply by $a_{id'}$ such that we only get a value on the left hand side for those containers that have an earlier destination port than d. If we instead just sum over these containers the cut gets stronger since we increase the left hand side by adding more terms to it. We have to divide the equation into two cases though because if there are two 20-foot containers in the cell the sum will be two. So let T' and F' be the two sets that only contains the $20'$ and $40'$ containers which have an earlier destination port than d respectively. The cut is then the two equations

$$\begin{aligned} \frac{1}{2}\sum_{i \in T'} c_{jk'i} &\leq \delta_{jkd} \qquad \forall j\, \forall k\, \forall d\, \forall k' \\ \sum_{i \in F'} c_{jk'i} &\leq \delta_{jkd} \qquad \forall j\, \forall k\, \forall d\, \forall k' \end{aligned} \tag{19}$$

where $k' \in \{1, \dots, k-1\}$. Another way to add more terms to the left hand side, and create a cut that will remove even more of the solution area, is to sum over the cells underneath k

$$\sum_{k'=1}^{k-1} a_{id'} c_{jk'i} \leq \delta_{jkd} \qquad \forall j\, \forall k\, \forall i\, \forall d\, \forall d' \tag{20}$$

where $d' \in \{2, \dots, d-1\}$. By including (19) and (20) instead of (17) the algorithm should become faster because the solution area for the LP-relaxation has been confined.[4]

7 Results

The program creates an output file with the values for the variables that appear in the optimal solution. An example of one can be seen in Appendix H where the data for location 55 in bay 14 was used. The optimal solution for the objective function is not directly given in this file but it can be read from the CPLEX outprint, see Appendix I. We have been given data sets for 9 different locations in a container ship and the optimal solutions found with our program are listed in Table 13. We can see in all the locations there are no empty stacks because the penalty for the term for

Bay	Location	Number of stacks	A	B	C	D	Total objective value
3	11	4	400	800	0	0	1200
4	15	5	500	1000	0	0	1500
8	30	4	400	1000	900	0	2300
9	34	4	400	800	0	0	1200
12	46	4	400	800	0	0	1200
12	48	10	1000	2600	0	0	3600
14	55	5	500	1600	50	0	2150
14	56	10	1000	2800	2150	0	5950
15	60	10	1000	2200	100	0	3300

Table 13: Results for running the program for the different input data. The values in column A, B, C and D are the penalties for empty stacks, pure stacks, non-reefer containers in reefer cells and overstowing respectively.

empty stacks is exactly 100 times the number of stacks. In location 8.30, 12.48, 14.55, and 14.56 all the stacks are not pure because the penalties in column B is greater than 200 times the number of used stacks. In the rest of the locations the penalties for pure stacks are at their minimum value hence all stacks are pure. The penalty for overstowing is zero

[4]The program was made with the two cuts so we added the original equation ourselves, see Appendix F.

in all the locations, this is due to it has the highest coefficient in the objective function so it will be minimized before the others.

7.1 Trade-off

Minimizing overstowing to zero can have caused that one of the other terms not have been minimized optimally so we will investigate this by running the program with each of the objectives.

The results from running the program with a single objective term are in Table 14 compared with the value they get when the program is run with the complete objective function with all the different terms included. They have the same value in both cases so no trade-off have occurred here.

Our data sets are created in such a way that the number of loaded containers and those to be loaded fill up every single cell and stack in the location. It will therefore neither be possible to get any empty stacks nor can the number of non-reefer containers in reefer cells be changed because it will always be the number of reefer cells that are not used for a reefer container. So no matter what order we minimize the objectives in the penalties from these two objectives will always be the same hence we can not test if any trade-off in general happen between them and minimizing overstowing.

If all stacks are pure there will not either be any overstowing. So when both the penalties for overstowing and non pure stacks are minimized and overstowing is prioritized highest, the number of pure stacks will also be optimal because it does not prevent overstowing from reaching it's optimal value. We can therefore conclude that no trade-off occur between pure stacks and minimizing overstowing.

Objective term	Value 1	Value 2
Pure stacks	2200	2200
Empty stacks	1000	1000
Reefer	100	100
Overstowing	0	0

Table 14: Results for location 60 in bay 15 where value 1 is the value of the objective function when the program is run with only one objective term included at the time and value 2 is the term's contribution to the objective value when all terms are included in the objective function.

7.2 Effect of cuts

In the manual preprocess we changed the equation for δ_{jkd} and to see the effect of this we will run the program with different combinations of the original equation and the cuts. First we run the program with each of the three different equations, then the program with the original equation and one cut at the time, and finally we run a program with all three equations. These results for location 55 in bay 14 and the result from Table 13 where the program was run with both cuts are shown in Table 15. In all the cases the optimal solution was found but not proved in some because in

Constraint	Objective	Time 1	Time 2	At node	Nodes left
(19)&(20)	2150	16.60	5749	44912	0
(17)	2150	25.56	188802[1]	758000	72712
(19)	2150	41.25	2439	28303	0
(20)	2150	27.15	54683[1]	428700	31267
(17)&(19)	2150	8.20	49225[1]	539100	83205
(17)&(20)	2150	5.43	90864[1]	1469200	121088
(17)&(19)&(20)	2150	16.92	56699[1]	577100	60509

Table 15: Time 1 denotes the time in seconds it takes for the program to find the objective value and Time 2 the time it in seconds takes for the program to prove that the objective found in Time 1 actually is the optimal value. [1]Indicates that the program was stopped after the given time even though it had not proved optimality of the objective found at Time 1.

these the number of nodes that still had to be checked kept increasing and the program became slower and slower so we stopped these. In the last two columns in Table 15 the number of nodes that has been solved and those left to investigate respectively are listed and as can be seen the programs we stopped still had many nodes left to check.

We can clearly see an effect of the cuts and for this location the program is actually fastest with just cut (19) on it's own (Time 2 in Table 15), but it takes longer time to initially find the optimal solution than when both cuts are included (Time 1). The number of nodes that has been examined before optimality has been proved is also smallest when only cut (19) is included. But in general there is neither any connection between Time 1 and Time 2 nor between the nodes investigated and the time used for it.

If the program is run with only cut (20), with the original equation on it's own or in any combination with the cuts it becomes much slower. Since we did not run those programs to the end we do not know how much slower but the one we ran for the longest time was already about 77 times slower than the fastest program and it was far from finished.

The program run with the other data sets was solved very fast except one, so they could not be used to further investigate the effect of the cuts. Location 48 in bay 12 was on the other hand extremely slow we therefore tried to run it with only cut (19) to see if it would improve the time for this location as well. In Table 16 we can see it is solved in less than half the time than with both cuts included.

Constraint	Objective	Solution time	Nodes investigated
(19)&(20)	3600	68	40302
(20)	3600	25	54045

Table 16: Effect on the solution time in hours for solving location 48 in bay 12 when a cut is removed from the program.

8 Comparing the methods

In integer programming the branch and bound method sets a lower bound for how low the IP solution can get. When an IP solution has been found we can in the gap column in the CPLEX outprint, see Appendix I, follow how far the lower bound is from the current IP solution. The gap will eventually be zero and when it happens optimality has been proved but we might not wait for it to happen and just stop the algorithm earlier if we consider the gap to be low enough. Of the three methods linear programming is the only one that proves optimality, so in Table 17 we have given two times, one for when the optimal solution is first found, and one for when it is proved, but for the fastest problems we can not see a difference in the two times in the CPLEX outprint.

Bay	Location	Objective	Time 1, sec	Time 2, sec
3	11	1200	<0.82	<0.82
4	15	1500	<0.93	<0.93
8	30	2300	<0.83	<0.83
9	34	1200	<2.34	2.34
12	46	1200	<4.85	4.85
12	48	3600	214.21	244285.00
14	55	2150	16.60	5749.00
14	56	5950	<4.08	4.08
15	60	3300	<0.99	<0.99

Table 17: Results for the different input data. Time 1 denotes the time it takes for the program to find the objective value and Time 2 the time it takes for the program to prove that the objective found in Time 1 actually is the optimal value.

Constraint programming and local search are methods that look for feasible solutions and keep searching in hope of finding a better solution. Neither of the methods has as integer programming a measure for how far the found solution is from the optimal solution and they can not prove if it is the optimum. But as we did in Chapter 7 common sense can be used to set a theoretical lower bound for the problem. In fact with our data sets the optimal solution is not far from it, but with other data sets it might not be the case and then the theoretical lower bound will not be a good indicator for how good the found solution is.

Local search can often quickly find a feasible solution but it can take long time before it finds the optimal solution and it might never reach it. The results from this method in Table 18 is found using the hill-climbing algorithm with sideways moves and random restart strategy. The algorithm is run five times and the best solution is then selected. In

Bay	Location	Objective	Time sec.	Gap %
3	11	1500	1.00	20.0
4	15	1500	2.49	0.0
8	30	2300	1.08	0.0
9	34	1200	0.82	0.0
12	46	1200	0.87	0.0
12	48	6200	4.29	35.7
14	55	2850	2.48	23.2
14	56	5950	8.86	0.0
15	60	3760	4.55	12.2

Table 18: Results for the container stowage problem solved with the local search algorithm by Dario Pacino (from the IT University in Copenhagen). The gap is how far the solution is from the optimal solution.

five of the nine locations the local search algorithm finds the optimal solution and the rest are up till 35.7 percent off the optimal solution. In Table 19 the results found with constraint programming is listed and the optimal solution is found in all locations. To find these a constraint solver called Gecode is used, it is similar to CPLEX but instead of IPs it solves CSPs. After the first solution is found constraints can be added or symmetries broken in a stack to improve the result.

Bay	Location	Objective	Time sec.	Gap %
3	11	1200	0.01	0.0
4	15	1500	0.06	0.0
8	30	2300	0.80	0.0
9	34	1200	4.17	0.0
12	46	1200	2.56	0.0
14	55	2150	1.82	0.0
14	56	5950	0.02	0.0
15	60	3300	0.37	0.0

Table 19: Results for the container stowage problem solved with constraint programming by Alberto Delgado-Ortegon (from the IT University in Copenhagen). The gap is how far the solution is from the optimal solution.

As the results show constraint programming and local search can find the optimal solution but they can not as integer programming prove that it actually is the optimum, so the time it takes to run the programs is not comparable. If anything should be compared Time 1 in Table 17 should be used for integer programming, but it is still not a proper comparison especially not since CPLEX does not provide us with exact times so a great uncertainty is connected with them. Constraint programming and local search are run at a big powerful server where for integer programming we have just used normal computers which makes a big difference so it contributes further to that it is not that useful to compare the times. Even with these differences we can see that in almost all cases local search uses the longest time to find a solution that not even always is optimal.

9 Discussion

In this project we have created our own model for finding optimal solutions to an under deck subproblem of the container stowage problem. We started with a simple model and subsequently added more details by removing some of the simplifications until we ended up with a useful model. In our model we only considered a section of the ship at the time so if it should be applied to the whole ship it must be ensured that the weight is evenly distributed. But in the data sets we had it was already decided which containers had to be placed in which location, so one should assume this was taken into account under the construction of the data sets.

After the model was implemented in CPLEX the optimal solutions was found for the given locations. The program was then run with only some of the objectives to see if any trade-off had occurred. We found that minimizing overstowing did not cause fewer pure stacks but we could not conclude if any trade-off happened between overstowing and any of the other objectives because of the construction of our data sets. However it is most likely that some trade-off do happen, one could for example imagine that instead of placing all containers in as few stacks as possible more stacks would be used to avoid overstowing.

Before implementing the model we created some cuts to make the final program faster and the effect of these was examined for one location. We could see they were really effective and that one cut was better than the other, but again our data sets were not sufficient because all locations but one were solved so fast that we could not see any changes there. For some it even seemed like the opposite cut of what we found was fastest. We could therefore not conclude anything beside the cuts do change the fastness of the program and that maybe it depends on the problem which combination of the cuts is best.

At the end the results from integer programming, constraint programming, and local search was compared. We knew that integer programming would find and prove the optimal solution, it was just a matter of how long time it would take. Local search seemed to be the worst method for solving these problems, but on the other hand we did not expect it to find the optimal solution in so many cases. That constraint programming found the optimal solution so fast was surprising because it is not always that efficient for optimization problems, so here a good heuristic must have been found to improve the method.

This project was part of a bigger comparison of the tree solution methods for solving optimization problems. It is not possible to pick one as the best method because they do different things and each have their own advantages. So what this project also will be used to by others is to try to find a combination of the methods to make a better and faster program. The problem with the branch and bound method in constraint programming is that the efficiency of it is determined by how good a heuristic function is used and how early a good bound is found. A way to improve it could then be to provide an initially good bound from one of the other solution methods such that more branches can be pruned by optimality hence the search of the whole tree is finished faster.

9.1 Further work

If we should continue this project we would like to check if our alternative model with fewer variables in Section 5.3.1 would make the program faster. It could also be very interesting to get more varied data sets, especially some that does not fill up the whole location such that we further could investigate trade-off and the effect of cuts and maybe reach a better understanding of how the model works. Unfortunately we got the program with the implementation of our model quite late, so we did neither have enough time to make all the changes we wanted nor to run the necessary test for analyzing the changes. One of the main things we also would have done was to define a function that prints the exact time for when the optimal solution is reached and one for when the program is finished instead of just for every thousand node.

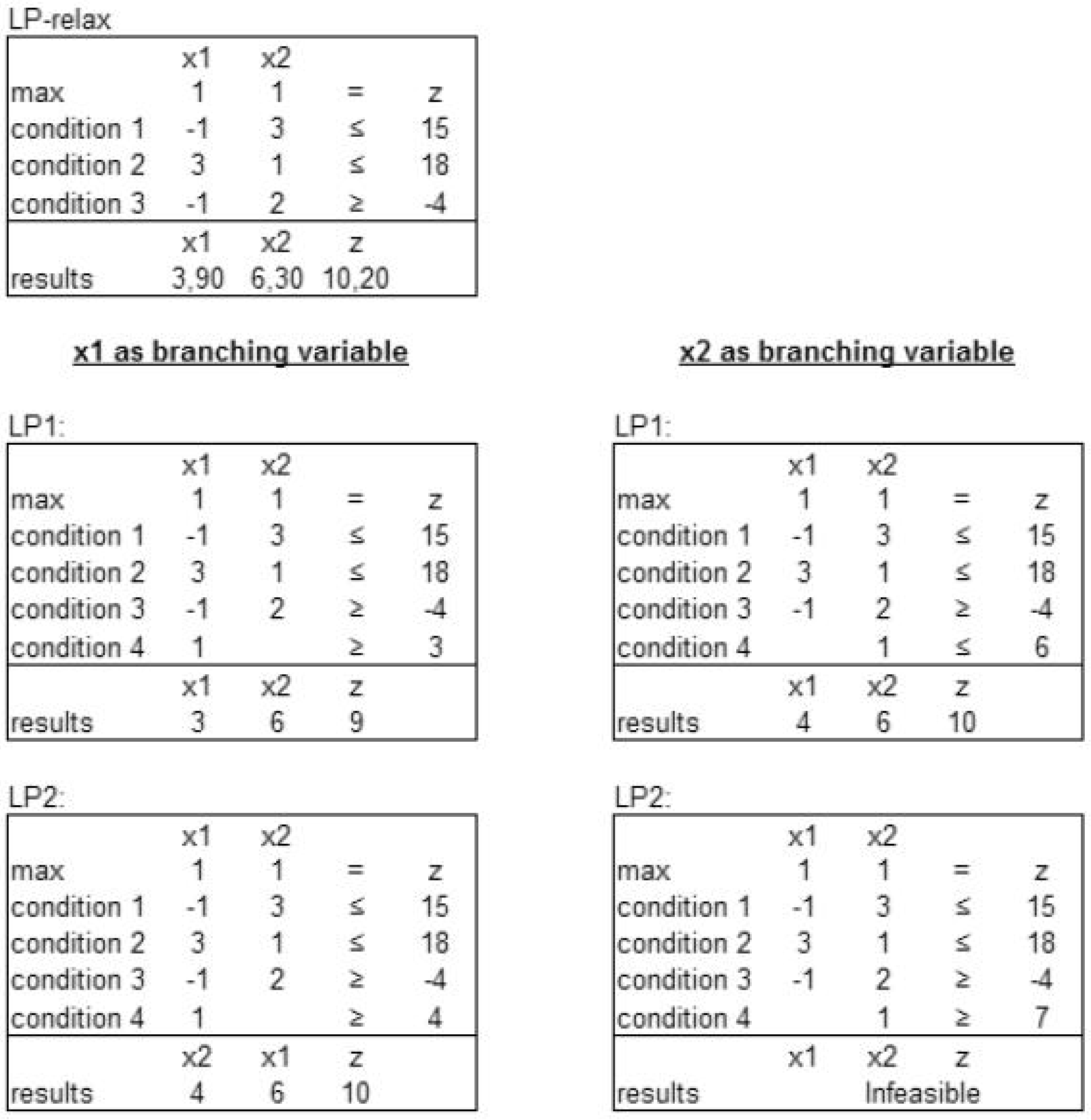
We have used CPLEX 9.1 but there exist a updated version CPLEX 11.0 which is most likely to be faster. As mentioned in Chapter 8 the computer capacity also has an impact on how fast the program finds the optimal solution, so it would be interesting to see how much the time for our program could be improved with a more powerful computer and the newest version of CPLEX.

Another obvious continuation to this project would be to try to make the model more complete and thereby reflect the real world problem better. The model could be extended by removing the restrictions and assumptions we have made during the modelling process. We could for example change our variable for a cell to indicate if the container is placed in the fore or aft position of the cell. More of the container types mentioned in Section 3.1 can also be included and since IMO containers appear frequently and it does not seem too difficult to include them, they would be a good choice to start with.

We have only one loading port in our model so in every port after unloading containers the program has to be run with updated data sets about which containers there still are on board and which new ones there has to be loaded from the port. If the data sets for more ports are available it could be interesting to try to extend the model to include all loading ports such that the program only have to be run in the first port and give an optimal solution for each of the following ports.

In this project we did not build a complete model for the under deck container stowage problem neither did we have any intention of this. As explained earlier it might not even be possible to achieve a complete model, but there are still many extensions and improvements that can be done.

A: Excel sheet



LP-relax

	x1	x2		
max	1	1	=	z
condition 1	-1	3	≤	15
condition 2	3	1	≤	18
condition 3	-1	2	≥	-4
	x1	x2	z	
results	3,90	6,30	10,20	

x1 as branching variable

LP1:

	x1	x2		
max	1	1	=	z
condition 1	-1	3	≤	15
condition 2	3	1	≤	18
condition 3	-1	2	≥	-4
condition 4	1		≥	3
	x1	x2	z	
results	3	6	9	

LP2:

	x1	x2		
max	1	1	=	z
condition 1	-1	3	≤	15
condition 2	3	1	≤	18
condition 3	-1	2	≥	-4
condition 4	1		≥	4
	x2	x1	z	
results	4	6	10	

x2 as branching variable

LP1:

	x1	x2		
max	1	1	=	z
condition 1	-1	3	≤	15
condition 2	3	1	≤	18
condition 3	-1	2	≥	-4
condition 4		1	≤	6
	x1	x2	z	
results	4	6	10	

LP2:

	x1	x2		
max	1	1	=	z
condition 1	-1	3	≤	15
condition 2	3	1	≤	18
condition 3	-1	2	≥	-4
condition 4		1	≥	7
	x1	x2	z	
results		Infeasible		

B: The GAMS program

```
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m

sets             j       stacks  /1*2/
                 k1      cells in stack 1 /1*3/
                 k2      cells in stack 2 /1*2/
                 ;


scalars          L_20    number of 20’ to be loaded /8/
                 L_40    number of 40’ to be loaded /1/
                 ;

variables        t_jk1(k1)  number of 20’ in cell 1k1
                 t_jk2(k2)  number of 20’ in cell 2k2
                 f_jk1(k1)  number of 40’ in cell 1k1
                 f_jk2(k2)  number of 40’ in cell 2k2
                 d_jk1(k1)  if there is none or some 20’ in cell 1k1
                 d_jk2(k2)  if there is none or some 20’ in cell 2k2
                 sI         total number of containers at the ship
                 z          object
                 ;

binary variables f_jk1, f_jk2, d_jk1, d_jk2;
integer variable t_jk1, t_jk2, sI;

t_jk1.up(k1)=2;
t_jk2.up(k2)=2;

alias(k1,m1);
alias(k2,m2);

equations        object          define object fct
                 load20          all 20’ has to be loaded
                 load40          all 40’ has to be loaded
                 totI            define sI
                 delta1g(k1)     defining d_jk1
                 delta1l(k1)     defining d_jk1
                 delta2g(k2)     defining d_jk2
                 delta2l(k2)     defining d_jk2
                 placefk1(k1,m1) the cell underneath should be full
                 placefk2(k2,m2) the cell underneath should be full
                 placetk1(k1,m1) the cell underneath should be full
                 placetk2(k2,m2) the cell underneath should be full
                 eitherk1(k1)    either 1 40’ or 1 or 2 20’ in a cell
                 eitherk2(k2)    either 1 40’ or 1 or 2 20’ in a cell
                 ;

object ..              z=e=sI;
load20 ..              L_20 =e=sum(k1,t_jk1(k1))+sum(k2,t_jk2(k2));
load40 ..              L_40 =e=sum(k1,f_jk1(k1))+sum(k2,f_jk2(k2));
totI ..                sI =e= sum(k1,t_jk1(k1))+sum(k2,t_jk2(k2))+
                       sum(k1,f_jk1(k1))+sum(k2,f_jk2(k2));
delta1g(k1) ..         t_jk1(k1)-d_jk1(k1)=g=0;
delta1l(k1) ..         t_jk1(k1)-2*d_jk1(k1)=l=0;
delta2g(k2) ..         t_jk2(k2)-d_jk2(k2)=g=0;
delta2l(k2) ..         t_jk2(k2)-2*d_jk2(k2)=l=0;
placefk1(k1,m1)$
(ord(m1) lt ord(k1)).. 2*f_jk1(m1)+t_jk1(m1)-2*f_jk1(k1)=g=0;
placefk2(k2,m2)$
(ord(m2) lt ord(k2)).. 2*f_jk2(m2)+t_jk2(m2)-2*f_jk2(k2)=g=0;
placetk1(k1,m1)$
(ord(m1) lt ord(k1)).. 2*d_jk1(k1)-t_jk1(m1)=l=0;
placetk2(k2,m2)$
```

```
(ord(m2) lt ord(k2)).. 2*d_jk2(k2)-t_jk2(m2)=l=0;
eitherk1(k1) ..        f_jk1(k1)+d_jk1(k1)=l=1;
eitherk2(k2) ..        f_jk2(k2)+d_jk2(k2)=l=1;

model ship /all/;
solve ship using mip maximizing z;

display z.l, t_jk1.l, f_jk1.l, t_jk2.l, f_jk2.l;
```

C: GAMS results

```
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n

----     84 VARIABLE z.L        =        9.000  object

----     84 VARIABLE t_jk1.L  number of 20’ in cell 1k1

         1 2.000,    2 2.000


----     84 VARIABLE f_jk1.L  number of 40’ in cell 1k1

         3 1.000


----     84 VARIABLE t_jk2.L  number of 20’ in cell 2k2

         1 2.000,    2 2.000


----     84 VARIABLE f_jk2.L  number of 40’ in cell 2k2

         ( ALL       0.000 )


EXECUTION TIME       =        0.000 SECONDS
```

D: Data file for bay 14

The first line gives the number of:
destinations, containers to load, containers loaded, stacks, cells, locations and tiers.
After `#POD` the destination ports are given.
After `#LOCATIONS` is which locations in the bay the containers has to be loaded in.
Under `#CONTAINERS TOLOAD` and `#CONTAINERS LOADED` the columns give the containers:
stack ID, cell ID, position in the cell(-1: a 20' in fore, 0: a 40', 1: a 20' in aft), weight, height, length, destination port, is reefer(0: no, 1: yes), in which location it should be/is placed.
So the first three columns are only for containers loaded, but here there are no loaded containers.
Under `#STACKS` the columns give the stacks: max weight, max height, location.
Under `#CELLS` the columns give details for the cells:
stack ID, reefer plug fore(0: no, 1: yes), reefer plug aft(0: no, 1: yes), capacity for 20' fore(0: no, 1: yes), capacity for 20' aft(0: no, 1: yes), capacity for 40'(0: no, 1: yes), location.

```
2 40 0 5 45 1 9
#POD
5 7
#LOCATIONS
55
#CONTAINERS_TOLOAD
0 0 0 12900.000000 2.895600 40 7 1 55
0 0 0 23300.000000 2.895600 40 5 1 55
0 0 0 24900.000000 2.895600 40 7 1 55
0 0 0 8000.000000 2.590800 40 5 0 55
0 0 0 15400.000000 2.895600 40 7 1 55
0 0 0 26400.000000 2.895600 40 5 1 55
0 0 0 23500.000000 2.895600 40 5 1 55
0 0 0 24400.000000 2.895600 40 7 1 55
0 0 0 13800.000000 2.895600 40 7 1 55
0 0 0 24200.000000 2.895600 40 7 1 55
0 0 0 15300.000000 2.895600 40 7 1 55
0 0 0 23100.000000 2.895600 40 5 1 55
0 0 0 23300.000000 2.895600 40 5 1 55
0 0 0 14900.000000 2.895600 40 7 1 55
0 0 0 23100.000000 2.895600 40 5 1 55
0 0 0 7500.000000 2.590800 40 5 0 55
0 0 0 23700.000000 2.895600 40 5 1 55
0 0 0 23500.000000 2.895600 40 5 1 55
0 0 0 23200.000000 2.895600 40 5 1 55
0 0 0 15400.000000 2.895600 40 7 1 55
0 0 0 24500.000000 2.895600 40 7 1 55
0 0 0 24000.000000 2.895600 40 5 1 55
0 0 0 23000.000000 2.895600 40 5 1 55
0 0 0 23100.000000 2.895600 40 5 1 55
0 0 0 25600.000000 2.895600 40 5 1 55
0 0 0 5900.000000 2.590800 40 5 0 55
0 0 0 24620.000000 2.895600 40 7 1 55
0 0 0 24600.000000 2.895600 40 7 1 55
0 0 0 24600.000000 2.895600 40 7 1 55
0 0 0 15400.000000 2.895600 40 7 1 55
0 0 0 7900.000000 2.590800 40 5 0 55
0 0 0 24200.000000 2.895600 40 7 1 55
0 0 0 9600.000000 2.590800 40 5 0 55
0 0 0 24700.000000 2.895600 40 7 1 55
0 0 0 24300.000000 2.895600 40 7 1 55
0 0 0 9800.000000 2.590800 40 5 0 55
0 0 0 25300.000000 2.895600 40 7 1 55
0 0 0 24700.000000 2.895600 40 7 1 55
0 0 0 24300.000000 2.895600 40 5 1 55
0 0 0 24400.000000 2.895600 40 7 1 55
#CONTAINERS_LOADED
#STACKS
420000.000000 23.800000 55
420000.000000 23.800000 55
```

```
420000.000000 23.800000 55
420000.000000 23.800000 55
420000.000000 23.800000 55
#CELLS
1 0 1 1 1 1 55
1 0 1 1 1 1 55
1 0 1 1 1 1 55
1 0 1 1 1 1 55
1 0 1 0 0 1 55
1 0 1 0 0 1 55
1 0 1 0 0 1 55
1 0 0 0 0 1 55
1 0 0 0 0 1 55
2 0 1 1 1 1 55
2 0 1 1 1 1 55
2 0 1 1 1 1 55
2 0 1 1 1 1 55
2 0 1 0 0 1 55
2 0 1 0 0 1 55
2 0 1 0 0 1 55
2 0 0 0 0 1 55
2 0 0 0 0 1 55
3 0 1 1 1 1 55
3 0 1 1 1 1 55
3 0 1 1 1 1 55
3 0 1 1 1 1 55
3 0 1 0 0 1 55
3 0 1 0 0 1 55
3 0 1 0 0 1 55
3 0 0 0 0 1 55
3 0 0 0 0 1 55
4 0 1 1 1 1 55
4 0 1 1 1 1 55
4 0 1 1 1 1 55
4 0 1 1 1 1 55
4 0 1 0 0 1 55
4 0 1 0 0 1 55
4 0 1 0 0 1 55
4 0 0 0 0 1 55
4 0 0 0 0 1 55
5 0 1 1 1 1 55
5 0 1 1 1 1 55
5 0 1 1 1 1 55
5 0 1 1 1 1 55
5 0 1 0 0 1 55
5 0 1 0 0 1 55
5 0 1 0 0 1 55
5 0 0 0 0 1 55
5 0 0 0 0 1 55
```

E: The header fil, read.h

```
#include "cplex.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*************************************************************************
   The data structure Size has the following attributes:
   PODs:        Number of discharge ports for the containers on board and
                to be loaded in the bay.
   conToLoad:   Number of containers to load
   contLoaded:  Number of containers already loaded in the ship.
   stacks:      Number of stacks in the bay.
   tiers:       Number of tiers in the bay.
   cells:       Number of cells in the bay.
   locations:   Number of locations in the bay.
 *************************************************************************/

typedef struct sizes{
  int PODs;
  int contToLoad;
  int contLoaded;
  int stacks;
  int tiers;
  int cells;
  int locations;} Sizes;

/*************************************************************************
  The data structure Cont storages the info of a container. The first
  three attributes are set to 0 in case the container hasn't been loaded yet.
  stackID   : Stack where the container is storaged, if it has been already
              loaded
  cellID    : Tier where the container is storaged.
  pos       : -1 represents the fact that the container is storaged at the
              fore part of the cell,  0 that is a 40 foot cont and 1 that
              the container is storaged at the aft part.
  weight    : The weight of the container in kg.
  height    : height of the container in m.
  length    : length of the container,  either 20 or 40 foot.
  dsp       : Discharge port of the container.
  is_reefer : whether the container is reefer or not.
  location  : location where the container is/should be allocated
 *************************************************************************/

typedef struct cont{
  int stackID;
  int cellID;
  int pos;
  float weight;
  float height;
  int length;
  int dsp;
  int is_reefer;
  int location;} Cont;

/*************************************************************************
  Information about stacks:
  weight    :  max weight in kg allowed in the stack
  height    :  max height in m allowd in the stack
  location  :  location where the stack is placed
 *************************************************************************/

typedef struct stack{
  float weight;
  float height;
  int location;} Stack;

/*********************************************************************
  Information about cells:
  stackID       :  stack where the cell belongs to
  reeferFore    :  whether the fore part is reefer
  reeferAft     :  whether the aft part is reefer
  capFore       :  whether the fore part is availabe to store a cont
  capAft        :  whether the aft part is availabe to store a cont
  cap40         :  whether the cell is availabe to store a 40 cont
  location      :  location where the cell is located
 *********************************************************************/

typedef struct cell{
  int stackID;
  int reeferFore;
  int reeferAft;
  int capFore;
  int capAft;
  int cap40;
  int location;} Cell;

/*********************************************************************
This function reads the number of PODs, locations,  containers to load,
containers loaded,  stacks and cells of a bay from the input file.
Parameters:
  name :  name of the file where the date is stored.
  s    :  A pointer to a Sizes structure.
*********************************************************************/

void getSizes(const char *name, Sizes *s);

/*********************************************************************
This function reads all relevant data for the container
problem from the file with name "name".
*********************************************************************/

void read_containter_problem_data(char *name,int *PODs,int *Locations,
                                  Cont *contToLoad,Cont *contLoaded,
                                  Stack * stacks,Cell * cells,Sizes *s);

/*********************************************************************
Function that prints the objective function.
*********************************************************************/

void print_obj(FILE *out_file,Sizes *s,int *PODs,Cont *contToLoad,Cont
       *contLoaded,Stack *stacks,Cell *cells,int*size_each_stack);

/*********************************************************************
Function that prints all the constraints.
*********************************************************************/

void print_the_constraints(FILE *out_file,Sizes *s,int *size_each_stack,
                           Cell *cells,Cont *contToLoad,Cont *contLoaded,
                           Stack *stacks,int *PODs);

/*********************************************************************
Function that prints all the fixed variables.
*********************************************************************/

void print_fixed_variables(FILE *out_file,Sizes *s,Cont *contLoaded);

/*********************************************************************
Function that prints the name of all the variables.
*********************************************************************/
```

```
void print_the_variables(FILE *out_file,Sizes *s,int *PODs,
                         int *size_each_stack,Cell *cells);
```

F: Our changes in the program

```
*************************************************************************
********* Old set of inequalities "D variables" must satisfy. **********
*************************************************************************

void print_delta_def(FILE *out_file,Sizes *s,int *size_each_stack,
Cell *cells,Cont *contToLoad,Cont *contLoaded,
Stack *stacks,int *PODs)
{
   int d,j,k,i;
   int firstcontainer;
   int cell_index;
   int cell_index2;
   int toload;
   int loaded;
   int dtwo,ktwo;
   int firstterm;

   for(j=1;j<s->stacks+1;j++)
      for(d=2;d<s->PODs+1;d++)
        for(k=2;k<size_each_stack[j-1]+1;k++) {

   calc_cell_index(j,k,&cell_index,size_each_stack);

   if( cells[cell_index-2].capFore || cells[cell_index-2].cap40 ) {

firstterm=0;
   for(ktwo=1;ktwo<k;ktwo++)
      for(dtwo=1;dtwo<d;dtwo++)
    for(i=1;i<s->contToLoad+s->contLoaded+1;i++) {

calc_cell_index(j,ktwo,&cell_index2,size_each_stack);

                toload=0;
              loaded=0;
              if( i < s->contToLoad+1 )
         toload=1;
              else
         loaded=1;

      if( loaded &&
                          contLoaded[i-s->contToLoad-1].dsp == PODs[dtwo-1] ||
                          toload &&
                          contToLoad[i-1].dsp == PODs[dtwo-1] ) {

     if( (cells[cell_index2].capFore || cells[cell_index2].cap40) &&
                                 firstterm ) {
        fprintf(out_file,"C(%d,%d,%d) ",j,ktwo,i);
firstterm=0;
     }
     else if( cells[cell_index2].capFore || cells[cell_index2].cap40 )
                                fprintf(out_file,"+ C(%d,%d,%d) ",j,ktwo,i);
} }
 fprintf(out_file,"-%d D(%d,%d,%d) <= 0\n",2*(k-1),j,k,PODs[d-1]);
   }}}


*************************************************************************
**************** Contribution to objective from reefers.****************
*************************************************************************

   for(j=1;j<s->stacks+1;j++)
       for(k=1;k<size_each_stack[j-1]+1;k++) {
```

```
   calc_cell_index(j,k,&cell_index,size_each_stack);

   if( (cells[cell_index-1].capFore || cells[cell_index-1].cap40)
   && (cells[cell_index-1].reeferFore ||
       cells[cell_index-1].reeferAft) ){

 nreefs=cells[cell_index-1].reeferFore + cells[cell_index-1].reeferAft;

         for(i=1;i<s->contToLoad+s->contLoaded+1;i++) {

          toload=0;
          loaded=0;
          if( i < s->contToLoad+1 )
     toload=1;
          else
     loaded=1;

  if( toload && contToLoad[i-1].is_reefer ||
          loaded && contLoaded[i-1-s->contToLoad].is_reefer )

fprintf(out_file," - 50 C(%d,%d,%d) ",j,k,i);
      }
      if(nreefs==1)
fprintf(out_file,"+ 50 t(%d,%d) + 50 f(%d,%d)",j,k,j,k);
      else
fprintf(out_file,"+ 100 t(%d,%d) + 100 f(%d,%d)",j,k,j,k);
  } }
```

G: The main code, maersk.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <unistd.h>
#include <signal.h>
#include "etime.h"
#include "read.h"

int main(int argc, char *argv[]){

/************************************************************************
***SECTION THAT CONTAINS DECLARATIONS OF VARIOUS OBJECTS USED IN THE CODE ***
************************************************************************/
/* CPLEX structures. */
   CPXENVptr env;
   CPXLPptr  cpx_lp;

/* CPLEX parameter data. */
   int    paramintval;
   double paramdblval;

/* The file "input_name" is the file data from Alberto.
   The file "output_name" is the file to be given to CPLEX. */
   char *input_name = "inputBay5_19.txt";
   char *output_name = "inputBay5_19.lp";
   char *sol_name = "inputBay5_19.sol";

/* Structures for storing the data for the container problem. */
   Sizes *s;
   int *PODs;
   int * Locations;
   Cont *contToLoad;
   Cont *contLoaded;
   Stack *stacks;
   Cell *cells;

/* Vector that will contain the number of cells in each stack. */
   int *size_each_stack;

/* Counters in various "for" loops". */
   int d,i,j,k;

/* Pointer to the output file. */
   FILE *out_file;
   FILE *sol_file;

/* Integer to record the status returned from various functions. */
   int status;

/* Integer to contain the number of variables/columns. */
   int numcols;

/* Variables used to obtain names for the variables. */
   int surplus;
   int colnamespace;
   char **colname;
   char *colnamestore;

/* To record the integer solution returned by CPLEX. */
   double *x;

/************************************************************************
```

```
******SECTION FOR PRINTING THE PROBLEM INTO LP FORMAT FOR CPLEX TO READ*******
******************************************************************************/
/* Open the outputfile to which the containerproblem must be written in
   lp format. */
   out_file=fopen(output_name,"w");

/* Allocating memory for the structure that contains the size
   of the container problem (or exiting if not possible). */
   s = (Sizes *) malloc(sizeof(Sizes)*1);
   if( s == NULL ) {
      printf("Can't allocate memory!\n");
      exit(0);   }

/* Getting the size of the container problem. */
   getSizes(input_name,s);

/* Allocation of memory to the structures that will contain the
   information of the problem. */
   PODs = (int *) malloc(sizeof(int)*(s->PODs));
   Locations = (int *) malloc(sizeof(int)*s->locations);
   contToLoad = (Cont *) malloc(sizeof(Cont)*s->contToLoad);
   contLoaded = (Cont *) malloc(sizeof(Cont)*s->contLoaded);
   stacks= (Stack *) malloc(sizeof(Stack)*s->stacks);
   cells = (Cell *) malloc(sizeof(Cell)*(s->cells));
   size_each_stack = (int *) malloc(sizeof(int)*s->stacks);

/* Read all the data from the inputfile named "input_name". */
   read_containter_problem_data(input_name,PODs,Locations,contToLoad,
                             contLoaded,stacks,cells,s);

/* Calculate the size of each stack. Note that indexing starts at zero. */
   for(j=0;j<s->stacks;j++)
       size_each_stack[j]=0;
   for(j=0;j<s->stacks;j++)
       for(k=0;k<(*s).cells;k++) {
   if( cells[k].stackID == (j+1) )
      (size_each_stack[j])++;       }

/* Printing general problem characteristics to outputfile. */
   fprintf(out_file,"Problem\n");
   fprintf(out_file,"Container stowage\n");

/* Printing the objective function. */
   fprintf(out_file,"\nMinimize\n");
   fprintf(out_file,"obj : ");
   print_obj(out_file,s,PODs,contToLoad,contLoaded,stacks,
             cells,size_each_stack);

/* Printing the constraints. */
   fprintf(out_file,"\nSubject To\n");
   print_the_constraints(out_file,s,size_each_stack,cells,contToLoad,
                      contLoaded,stacks,PODs);

/* Printing the fixed variables. */
   fprintf(out_file,"\n Bounds\n");
   print_fixed_variables(out_file,s,contLoaded);

/* Printing the names of the variables
   (integer variables assumed binary by CPLEX). */
   fprintf(out_file,"\n Integer\n");
   print_the_variables(out_file,s,PODs,size_each_stack,cells);

/* Everything has been printed. */
   fprintf(out_file,"End\n");
```

```
/* Close the outputfile. */
   fclose(out_file);

/* Free allocated memory. */
   free(s);
   free(PODs);
   free(Locations);
   free(cells);
   free(stacks);
   free(contToLoad);
   free(contLoaded);
   free(size_each_stack);

   /*   return 0; */

/***************************************************************************
***SECTION FOR SOLVING THE PROBLEM WITH CPLEX GIVEN PROBLEM IN LP FORMAT***
***************************************************************************/
/* Open a CPLEX enviroment for loading and solving problems. If it is
   not possible, the program waits 5 seconds (sleeps) and tries again. */
   env = NULL;
      while( !env ) {
         env=CPXopenCPLEX(&status);
         if( !env ) {
            printf("CPLEX License unavailable %d... sleeping 5 seconds...\n",
                   status);
            sleep(5);         }  }

/* Sets a parameter so that CPLEX prints what it does on the screen. */
   CPXsetintparam(env, CPX_PARAM_SCRIND, 1);

/* Creates a "problem" in CPLEX which can later be
   referenced with the pointer "cpx_lp". */
   cpx_lp=NULL;
   cpx_lp=CPXcreateprob(env,&status,"Stowage problem");
   if( status || cpx_lp == NULL ) {
      printf("Could not create lp-problem\n");
      CPXcloseCPLEX(&env);
      return 0;   }

/* Reads a problem from a file in "lp" format at copies the
   result into the CPLEX problem "cpx_lp". */
   CPXreadcopyprob(env,cpx_lp,output_name,"lp");

/* Assigns the number of variables (or columns) of the problem
   to the integer "numcols". */
   numcols=CPXgetnumcols(env,cpx_lp);

/* Determines the amount of memory needed (the negative of "surplus")
   to store the column names. */
   status=CPXgetcolname(env,cpx_lp,NULL,NULL,0,&surplus,0,
        numcols-1);

/* If there is an error in the determination of the amount of
   space needed for variable names, we exit the program. */
   if (( status != CPXERR_NEGATIVE_SURPLUS ) &&
       ( status != 0 )                          )  {
      printf ("Could not determine amount of space for column names.\n");
      exit(0);   }

/* Assigns "colnamespace" the amount of memory needed. */
   colnamespace = - surplus;

/* Assigns physical memory for column names and exits if
   memory allocation is not possible. */
   colname      = (char **) malloc (sizeof(char *)*numcols);
   colnamestore = (char *)  malloc ((sizeof(char)*colnamespace));
   if ( colname      == NULL ||
        colnamestore == NULL   ) {
       printf ("Failed to get memory for column names.\n");
       exit(0);   }

/* CPLEX is called to get the actual variable (column) names. */
   status=CPXgetcolname(env,cpx_lp,colname,colnamestore,
colnamespace,&surplus,0,numcols-1);

/* If we fail to get the variable names, we exit the program. */
   if ( status ) {
       printf("CPXgetcolname failed.\n");
       exit(0);   }

/* Parameters for the CPLEX solver (see CPLEX manual online). */
   paramintval=3;
   CPXsetintparam(env,CPX_PARAM_VARSEL,paramintval);/*Use strong branching.*/
   paramintval=2;
   CPXsetintparam(env,CPX_PARAM_PROBE,paramintval);/*Use probing.*/
   /*    paramintval=1000;
 CPXsetintparam(env,CPX_PARAM_NODELIM,paramintval); */
   /* Stop after 1000 branch-and-bound nodes. */

/* Solve the integer program. */
   CPXmipopt(env,cpx_lp);

/* Allocates memory to the solution vector and exits if
   memory allocation is not possible. */
   x = (double *) malloc (numcols*sizeof(double));
   if( x == NULL ) {
      printf("No memory for solution.\n");
      exit(0);   }

/* Find the optimal solution and exit if this is not possible. */
   status=CPXgetmipx (env,cpx_lp,x,0,numcols-1);
   if( status ) {
      printf("Failed to obtain primal solution.\n");
      exit(0);   }

/* Open the solution file to which the solution found is written. */
   sol_file=fopen(sol_name,"w");

/* Printing an explaination of what the variables mean. */
   fprintf(sol_file,"C(j,k,i)=1 means container i is placed in
   stack j at position k.\n");
   fprintf(sol_file,"O(j,k)=1 means that position k in stack j
   is overstowing.\n");
   fprintf(sol_file,"P(j,d)=1 means that in stack j, the
   destination port d is present\n");
   fprintf(sol_file,"D(j,k,d)=1 means there is a container in stack
   j below position k with a destination port earlier than
   destination port d\n");
   fprintf(sol_file,"f(j,k)=1 means there is a forthy foot container
   at position k in stack j\n");
   fprintf(sol_file,"t(j,k)=1 means there are two twenty foot
   containers at position k in stack j\n");
   fprintf(sol_file,"E(j)=1 means that stack j is not empty\n\n");

/* Print all variables with the variable one and their
   name into the solution file. */
   for(j=0;j<numcols;j++) {
       if ( x[j] > 0.01 )
         fprintf (sol_file,"%-16s:  %lf\n", colname[j],x[j]);}
```

```
/* Close the solution file. */
   fclose(sol_file);

/* Close CPLEX down and end the program. */
   CPXcloseCPLEX(&env);

/* Free allocated memory. */
   free(colname);
   free(colnamestore);
   free(x);
   return 0;}
```

H: Output file for location 55 in bay 14

```
C(j,k,i)=1 means container i is placed in stack j at position k.
O(j,k)=1 means that position k in stack j is overstowing.
P(j,d)=1 means that in stack j, the destination port d is present
D(j,k,d)=1 means there is a container in stack j below position k
with a destination port earlier than destination port d
f(j,k)=1 means there is a forthy foot container at position k in stack j
t(j,k)=1 means there are two twenty foot containers at position k in stack j
E(j)=1 means that stack j i not empty
P(1,5)          :  1.000000      C(3,5,5)        :  1.000000
P(2,5)          :  1.000000      C(2,7,6)        :  1.000000
P(3,5)          :  1.000000      C(4,7,7)        :  1.000000
P(3,7)          :  1.000000      C(4,5,8)        :  1.000000
P(4,5)          :  1.000000      C(4,1,9)        :  1.000000
P(4,7)          :  1.000000      C(5,7,10)       :  1.000000
P(5,5)          :  1.000000      C(4,3,11)       :  1.000000
P(5,7)          :  1.000000      C(2,3,12)       :  1.000000
E(1)            :  1.000000      C(2,5,13)       :  1.000000
E(2)            :  1.000000      C(4,2,14)       :  1.000000
E(3)            :  1.000000      C(2,2,15)       :  1.000000
E(4)            :  1.000000      C(5,8,16)       :  1.000000
E(5)            :  1.000000      C(1,7,17)       :  1.000000
C(2,6,26)       :  1.000000      C(1,1,18)       :  1.000000
f(1,1)          :  1.000000      C(2,4,19)       :  1.000000
f(1,2)          :  1.000000      C(3,7,20)       :  1.000000
f(1,3)          :  1.000000      C(5,4,21)       :  1.000000
f(1,4)          :  1.000000      C(1,6,22)       :  1.000000
f(1,5)          :  1.000000      C(1,2,23)       :  1.000000
f(1,6)          :  1.000000      C(1,5,24)       :  1.000000
f(1,7)          :  1.000000      C(1,4,25)       :  1.000000
f(1,8)          :  1.000000      C(3,3,27)       :  1.000000
f(2,1)          :  1.000000      C(4,6,28)       :  1.000000
f(2,2)          :  1.000000      C(3,4,29)       :  1.000000
f(2,3)          :  1.000000      C(3,6,30)       :  1.000000
f(2,4)          :  1.000000      C(2,8,31)       :  1.000000
f(2,5)          :  1.000000      C(3,2,32)       :  1.000000
f(2,6)          :  1.000000      C(3,8,33)       :  1.000000
f(2,7)          :  1.000000      C(5,2,34)       :  1.000000
f(2,8)          :  1.000000      C(5,6,35)       :  1.000000
f(3,1)          :  1.000000      C(1,8,36)       :  1.000000
f(3,2)          :  1.000000      C(5,1,37)       :  1.000000
f(3,3)          :  1.000000      C(5,3,38)       :  1.000000
f(3,4)          :  1.000000      C(2,1,39)       :  1.000000
f(3,5)          :  1.000000      C(5,5,40)       :  1.000000
f(3,6)          :  1.000000      D(1,2,7)        :  1.000000
f(3,7)          :  1.000000      D(1,3,7)        :  1.000000
f(3,8)          :  1.000000      D(1,4,7)        :  1.000000
f(4,1)          :  1.000000      D(1,5,7)        :  1.000000
f(4,2)          :  1.000000      D(1,6,7)        :  1.000000
f(4,3)          :  1.000000      D(1,7,7)        :  1.000000
f(4,4)          :  1.000000      D(1,8,7)        :  1.000000
f(4,5)          :  1.000000      D(1,9,7)        :  1.000000
f(4,6)          :  1.000000      D(2,2,7)        :  1.000000
f(4,7)          :  1.000000      D(2,3,7)        :  1.000000
f(4,8)          :  1.000000      D(2,4,7)        :  1.000000
f(5,1)          :  1.000000      D(2,5,7)        :  1.000000
f(5,2)          :  1.000000      D(2,6,7)        :  1.000000
f(5,3)          :  1.000000      D(2,7,7)        :  1.000000
f(5,4)          :  1.000000      D(2,8,7)        :  1.000000
f(5,5)          :  1.000000      D(2,9,7)        :  1.000000
f(5,6)          :  1.000000      D(3,8,7)        :  1.000000
f(5,7)          :  1.000000      D(3,9,7)        :  1.000000
f(5,8)          :  1.000000      D(4,8,7)        :  1.000000
C(3,1,1)        :  1.000000      D(4,9,7)        :  1.000000
```

```
C(1,3,2)        :  1.000000       D(5,8,7)        :  1.000000
C(4,4,3)        :  1.000000       D(5,9,7)        :  1.000000
C(4,8,4)        :  1.000000
```

I: CPLEX outprint

```
MIP Presolve eliminated 4210 rows and 405 columns.
MIP Presolve modified 185 coefficients.
Aggregator did 10 substitutions.
Reduced MIP has 1515 rows, 1570 columns, and 16130 nonzeros.
Presolve time =    0.12 sec.
Probing added 3905 nonzeros
Probing time =    0.55 sec.
Clique table members: 5735
MIP emphasis: balance optimality and feasibility
Root relaxation solution time =    0.29 sec.

        Nodes                                       Cuts/
   Node  Left  Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0  1457.1429   221                   1457.1429      779
               1535.1443   195                   Cuts:  87     1720
               1621.4286   194                   Cuts:  76     2587
               1621.4286   181                   Cuts:  11     2726
               1621.4286   176                    Cuts:  2     2751
               1621.4286   170                    Cuts:  3     2858
               1621.4286   168                    Cuts:  4     2874
*    40+   15                0     2150.0000     1621.4286     4596   24.58%
    100    31  1831.6615    29     2150.0000     1737.0995     7289   19.20%
    200    50  1857.5261    54     2150.0000     1737.0995    11200   19.20%
    300    77     cutoff           2150.0000     1737.0995    15551   19.20%
    400   107  1950.0000     9     2150.0000     1746.2020    21397   18.78%
    500   125  1950.0000    44     2150.0000     1765.3637    24936   17.89%
    600   143  1829.3887    28     2150.0000     1791.4072    29363   16.68%
    700   156  1950.0000    12     2150.0000     1807.4319    34389   15.93%
    800   162  1829.3887    37     2150.0000     1829.3887    39386   14.91%
    900   191  1955.4520    23     2150.0000     1865.6709    43641   13.22%
   1000   204     cutoff           2150.0000     1886.5662    49413   12.25%
Elapsed time = 195.22 sec. (tree size =  0.17 MB)
   1100   220  1924.5477    56     2150.0000     1896.0423    54373   11.81%
   1200   244  1950.0000    13     2150.0000     1901.9531    57173   11.54%
   1300   254  1921.4286    82     2150.0000     1921.4286    64481   10.63%
   1400   285  1950.0000     9     2150.0000     1932.8756    70109   10.10%
   1500   293  1950.0000    23     2150.0000     1940.7792    79296    9.73%
   1600   298  2088.9054    55     2150.0000     1946.9264    90433    9.45%
   1700   306  1950.0000     8     2150.0000     1950.0000    98490    9.30%
   1800   307  1950.0000     4     2150.0000     1950.0000   106218    9.30%
   1900   317  1950.0000    12     2150.0000     1950.0000   113435    9.30%
   2000   320  1950.0000     8     2150.0000     1950.0000   120994    9.30%
Elapsed time = 367.53 sec. (tree size =  0.27 MB)

Elapsed time = 9033.43 sec. (tree size =  0.46 MB)
  44100   392  2053.9062     3     2150.0000     2038.3203  4570884    5.19%
  44200   338  2050.0000    38     2150.0000     2050.0000  4577327    4.65%
  44300   322  2050.0000    61     2150.0000     2050.0000  4583253    4.65%
  44400   330     cutoff           2150.0000     2050.0000  4588103    4.65%
  44500   314  2053.9062     3     2150.0000     2053.9062  4595351    4.47%
  44600   257     cutoff           2150.0000     2053.9062  4602778    4.47%
  44700   181     cutoff           2150.0000     2067.7604  4611288    3.83%
  44800   109     cutoff           2150.0000     2077.9297  4619385    3.35%
  44900    12     cutoff           2150.0000     2100.0000  4626671    2.33%

GUB cover cuts applied:  13
Clique cuts applied:  23
Gomory fractional cuts applied: 2
```

References

[1] Skibsfarten i tal-maj 2008, Danish homepage, Last visited august, 2008 (2008).
URL `http://www.danmarksrederiforening.dk/pdf/skibsfarten\_i\_tal\_052008.pdf`

[2] Transport-velstandens smøremiddel, Danish homepage, Last visited august, 2008 (2003).
URL `http://www.ebst.dk/publikationer/rapporter/godstransport/kap01.htm`

[3] History and largest ships, English homepage, Last visited august, 2008 (2006).
URL `http://en.wikipedia.org/wiki/Container\_ship`

[4] Jensen, R. M., Liner shipping: Containers, vessels, terminals and service networks, Unpublished (2007).

[5] Dansk selskab for operationsanalyse, Danish homepage, Last visited august, 2008 (2004).
URL `http://www.dorsnet.dk/or/index.php`

[6] Williams, H. P., *Model Building In Mathematical Programming*, Wiley, third edition (1994).

[7] Taha, H. A., *Operations Research, An Introduction*, Prentice-Hall, eighth edition (2007).

[8] Wolsey, L. P., *Integer Programming*, John Wiley & sons (1998).

[9] Russell, S. and Norvig, P., *Artificial Intelligence, A Modern Approach*, Prentice-Hall, second edition (2003).