ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Econometrics and OR

A flow-first route-next heuristic for liner shiping network design

---

Author:
Olivier Knappert
Student ID number:
449890

Supervisor:
Nemanja Milovanovic
Second assessor:
Wilco van den Heuvel

July 7, 2019

The views stated in this thesis are those of the author and not necessarily those of Erasmus School of Economics or Erasmus University Rotterdam.

Abstract

Designing cyclic sailing routes for a fleet of containers is a very complex problem. Due to the increasing awareness for environmental consequences, it has become more important to not only design cost-efficient, but also energy-efficient networks. This paper proposes a flow-first route-next heuristic, which seeks to maximize profit while stimulating so-called *slow steaming*.

This heuristic first creates a backbone flow which will be used to generate initial rotations. These initial rotations are improved using a Variable Neighborhood Search algorithm. Then, sailing speed is minimized such that all routes remain feasible. We perform this procedure two times, once allowing only simple cyclic (general) rotations and once allowing non-simple cyclic rotations, also referred to as *butterfly rotations*. In addition, the rotations are generated and improved in two manners, once assuming design speed and once assuming maximum speed.

Our results show that the revenue for flowing containers is higher when allowing for butterfly rotations. Also, when maximum speed is assumed while generating and improving rotations, the revenue is higher for both the general and butterfly algorithm. We see that after subtracting the cost for operating the rotations, no profitable solutions were found. However, the butterfly algorithm with maximum speed is the least loss-making algorithm.

Contents

1 Introduction

The liner shipping industry plays an essential part in the global economy. It has been a fast growing industry in the past few decades. Between 1983 and 2006 average containerized cargo growth per year surpassed the world gross domestic product growth per year by more than 5%. These were respectively 10% and 4.8% (Stopford, 2009). In 2015, total seaborne transported cargo was estimated to be 80% of the total global merchandise trade. This translates to over 10 billion tons of trade volumes in that year (UNCTAD, 2016).

Although the impact of container transport per ton-km is relatively low compared to other transport modes, it is still responsible for a big part of the total CO_2 emission. IMO (2014) estimated that in 2012, the shipping industry was responsible for 2.2% of the global CO_2 emission, of which 25% was caused by vessels. The awareness for environmental consequences nowadays is increasing, hence it is not only important to design cost-effective networks, but also energy-efficient networks.

Designing cyclic sailing routes for a designated fleet of container vessels that transports multiple comodities, also called The Liner Shipping Network Design Problem (LSNDP), is a very complex problem. Many factors exist which need to be taken into consideration when designing *services*. A *service* is a round trip sailed at a given frequency. In most cases weekly or bi-weekly service is assumed. This is done because it brings significant planning advantages for the stakeholders. LSNDP is comparable to the well known Vehicle Routing Problems (VRP), however research in the latter is much further developed. Brouer et al. (2014) believes that the lack of Operations Research (OR) within liner shipping is partly because of the barriers for new researchers to engage in the liner shipping research community. Constructing mathematical models and creating data for computational results requires profound knowledge of the domain and data sources. They discuss the domain of liner shipping and corresponding data well and present a benchmark suite which can be used to compare methods used to solve LSNDP. In this paper the same benchmark suite will be used to test the developed algorithms. Several papers have proposed methods to solve the LSNDP. In most of them a two-stage approach is applied in which first the routes of vessels is designed, and afterwards a multi-commodity flow problem (MCF) solved to flow containers over the existing routes.

In this paper a different approach will be covered from Pisinger (2016). Instead of first designing routes, and then using these routes to flow containers, we do the opposite. First, containers are flowed over a relaxed network and then routes are designed based on the container flows. A MCF model is solved to flow containers. After having generated several starting solutions, rotations are improved by means of a Variable Neighborhood Search (VNS) method. Finally, we minimize the sailing speed such that the solutions remain feasible. This algorithm is executed four times. We perform the before described procedure two times, once allowing only simple cyclic rotations (general) and once allow-

ing non-simple cyclic rotations, also referred to as butterfly rotations. In addition, the rotations are generated and improved in two manners, once assuming design speed and once assuming maximum speed. To test the algorithms proposed in this paper we make use of the Baltic instance of the *LINER-LIB 2012* benchmark suite.

We start by reviewing some previous works related to liner shipping problems in Section 2. Then, in Section 3 the construction of the initial rotations is explained into detail. In Section 4 we discuss the VNS used to find a best set of rotations. In Section 5 we explore the possibility of also allowing ships to call more than once at the same port on the same rotation. Section 6 describes a model which is used to minimize sailing speed after having optimized the rotations. Then, in Section 7 we will present and compare the results of the general algorithm versus the algorithm also allowing butterfly rotations. The algorithms described in this paper carry as fundamentals the algorithm proposed in Krogsgaard et al. (2018). Therefore, we will compare some of the results obtained with those of Krogsgaard et al. (2018). Finally, in Section 8 we will conclude our findings.

2 Literature Review

To get a good understanding of liner shipping network design problems Brouer et al. (2014) can be consulted. The paper gives a thorough explanation of the domain of liner shipping problems. They present a benchmark suite consisting of several data instances, which originate from world's largest shipping company, Maersk Line, and several other stakeholders. In addition, they present a useful formulation of a graph $G = (V, A)$ which can be used to help solving a MCF problem. In this formulation a port node is added for each rotation calling at the port. This enables the creation of transshipment arcs between two nodes and hence the incorporation of transshipment costs.

Alvarez (2009) also incorporated transshipments in its model, however the calculation of costs is less accurate. Alvarez uses one node per port. This prevents the model from detecting a transshipment when a rotation calls more than once at the same port. Hence, to prevent inaccurate costs Alvarez restricts the rotation to call at most once at each port. A disadvantage compared to Brouer et al. (2014), where butterfly rotations are allowed, which often leads to more cost-efficient rotations.

Both Brouer et al. (2014) and Alvarez (2009) solve the problems using a two-stage approach. They start by solving a MCF model, then rotations are improved based on the flow. Agarwal and Ergun (2008) present a mixed-integer linear model which incorporates several important constraints such as a weekly frequency and the possibility to transship cargo. They propose three algorithms to solve the MIP, namely a greedy heuristic, a column generation-based algorithm and a two-phase Benders decomposition-based algorithm.

Krogsgaard et al. (2018) presents a two-stage algorithm in which first a backbone flow is created. Thereafter, initial rotations are designed by means of a greedy construction heuristic. Then, these initial rotations are improved by means of a VNS.

We note that until now no exact solution methods have been found for solving large instances of LSNDP. Several algorithms have been invented, one with better computational results than the other. In Krogsgaard et al. (2018) the computational results of the algorithm used in that article are compared with the results of Brouer et al. (2014) and Brouer, Desaulniers and Pisinger (2014). In four out of six instances Krogsgaard managed to obtain better solutions. In addition, the time to find these solution was significantly shorter.

3 Construction Heuristic

3.1 Overview

As mentioned before, this paper considers a two-stage approach in which first an initial flow is constructed, which will be used to generate initial rotations. This initial flow is called the *backbone flow*. Subsection 3.2 covers the construction heuristic of this backbone flow. Then, Subsection 3.3 describes the greedy heuristic used to generate a set of feasible rotations. Lastly, in Subsection 3.4 we go more into detail regarding the MCF model used to flow demand and evaluate the rotations.

3.2 Constructing backbone flow

A simple heuristic is used in order to create a backbone flow. In the heuristic we make use of graph consisting of nodes en arcs. The nodes represent the ports and the directed arcs represent sailing waterways between a each pair of ports. This heuristic considers the economy of scale for flowing containers. This means that letting a ship sail a specific arc is expensive, but the cost per container decreases as the number of flowed containers over this edge increases. By cost we refer to the weighted distance between two ports. As we are dealing with an economy of scale we make use of a non-linear concave function $f_{ij}(x)$ to compute the weight of the distance between i and j. The weights w_{ij} are given by, $w_{ij} = f_{ij}(x_{ij} + 1) - f_{ij}(x_{ij})$, in which x_{ij} is the current flow on edge (i, j).

We start by calculating $f_{ij}(x_{ij})$ for every edge (i, j). First, the cost a_v of sailing arc (i, j) for each vessel class $v \in V$ is computed. a_v consists of bunkercosts and V is the set containing the six different vessel classes. Some information on the capacity and names of the different vessel classes can be found in Table 1 in Section 3.4.

Then, all points (a_v, u_v) are plotted in which u_v represents the capacity of vessel class v. Now, we approximate a square-root curve fitting using the 6 points representing the 6 different vessel classes. For each arc (i, j), b_{ij} and c_{ij} are estimated in the function, $f_{ij}(u) = b_{ij}\sqrt{u} + c_{ij}$. In Figure 1 an example is shown of six plotted points together with the approximated square-root function.

Having approximated $f_{ij}(x)$ for each arc, the heuristic is solved by repeatedly going through the following steps. In each iteration a shortest path is found for a container of a randomly chosen origin-destination pair with un-routed demand, also referred to as OD pair. The shortest path is found by solving the well known shortest path model (1)-(5) in CPLEX using the graph mentioned before, in which d_{ij} represents the length of arc (i, j). After having flowed a container, the weights of each edge are updated and the next container is flowed, until no un-routed demand is left.

Since, the manner in which containers are flowed strongly depends on the first several containers flowed, we repeat this procedure 10 times and take the average of the number

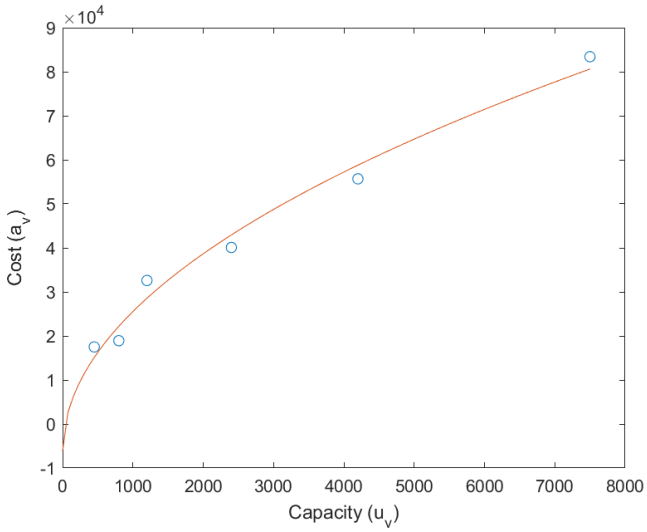

Figure 1: Example of approximated square-root function, $f_{ij}(u_v) = 1001.6\sqrt{u_v} - 6134.6$

of containers flowed over each arc. In Figure 2 an example is given of a backbone flow.

$$\min \sum_{(i,j)\in A} w_{ij} d_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0, \qquad \forall i \in N \setminus \{O, D\} \tag{2}$$

$$\sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = 1, \qquad i = O \tag{3}$$

$$\sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = -1, \qquad i = D \tag{4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A \tag{5}$$

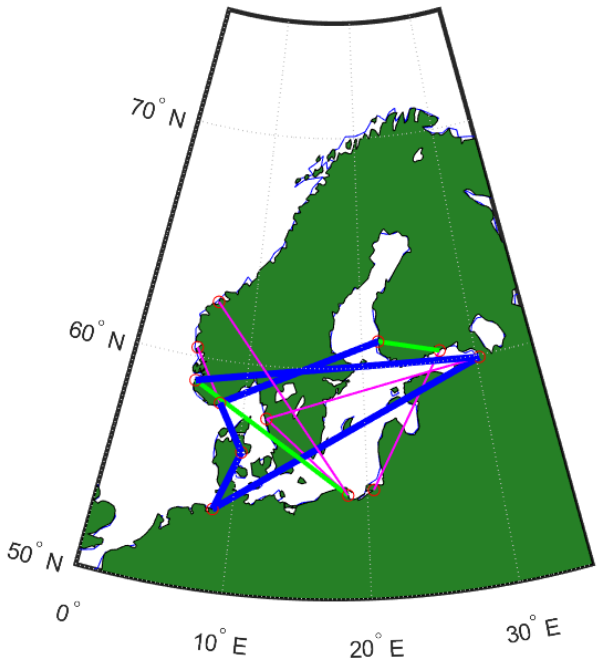

Figure 2: Typical backbone flow for the Baltic instance

3.3 Generating rotations

The second step in the two-stage approach is the generation of initial rotations. The idea of designing rotations based on the backbone flow is that we want to maximize the number of arcs with a large flow covered by a rotation. In this paper we consider a greedy heuristic which sequentially generates rotations. This means that the rotations are designed one at a time. A drawback of this approach is that the good arcs will not be distributed fairly among the rotations. However, our method ensures that arcs with a heavy load are assigned to a rotation of large vessels. The pseudocode for the greedy heuristic is given in Algorithm 1.

The generation of each rotation starts by finding the unserved arc with the biggest flow and adding a return arc. Afterwards, you repeatedly remove the return arc, find a new best arc, and add a new return arc to close the rotation. This new best arc is the arc with the biggest flow, either entering the port where the return arc entered, or leaving the port where the return arc left. We give a concrete example on which restrictions should hold when adding a new arc: both the new arc as well as the return arc have a flow smaller or equal to the capacity and are not yet served by another rotation; the total travel time plus the time to travel the new arc and return arc is smaller or equal than the maximum rotation time; the draft of the vessel is less or equal than the maximum draft of the two ports. Also, it is only allowed to call at each port. The time to sail arc (i, j) is given in days by $t_{ij}^v = \frac{d_{ij}}{24 * designSpeed_v}$. The design speed in nautical miles per hour differs per vessel, hence the index v. Arcs are being added as long as arcs exist satisfying the

above mentioned restrictions. If no such arc exists, the rotation generation terminates.

The total rotation time consists of the time needed to sail all arcs and the time spend in each port. For simplification it is assumed that each port stay has a duration of one day. Hence, to give an example we take a look at a rotation given by $\{0 \rightarrow 3 \rightarrow 5 \rightarrow 4 \rightarrow 0\}$ in which each number represents a port and the arrows an arc between a pair of ports. This rotation consists of four sailing arcs and 3 port calls at port 3, 5 and 4. However, as the vessel also has to load and unload when arriving at port 0 before starting the same rotation over, we also add a 24 hour port stay at port 0. Hence, total rotation time consists of the time to sail four arcs and four port stays.

Algorithm 1 Rotation generation algorithm

```
1: Initialization: Find unserved arc with biggest flow and add return arc to rotation r
2: while total travel time ≤ max. rotation time do
3:     Find new best arc_ij
4:     if arc_ij is feasible then
5:         Remove return arc from r, add arc_ij and new return arc to r
6:         Update total travel time
7:     else
8:         break
9: return r
```

For the VNS algorithm, next to be discussed in Section 4, we like to have several initial sets of rotations. Hence, we use 5 different backbone flows and for each flow we generate 20 sets of rotations. In each set the maximum rotation time for each vessel class is randomly chosen from the intervals shown in Table 1. In this table also the fleet for Baltic is shown. We aim for weekly frequency. This means that if a rotation takes 3 weeks, 3 vessels are needed to service this rotation. Using Table 1 we can deduce the number of rotations that can be generated. If for example in a specific set, the maximum rotation length for $Feeder_{450}$ equals 4 weeks, then we can generate at most 1 rotation.

We will do the entire rotation generation and Variable Neighborhood Search, later to be explained in Section 4, also assuming maximum speed of the vessels. This will probably lead to rotations covering more ports, hence better coverage of demand. However, as fuel consumption increases as speed increases, higher bunker cost is the consequence. Therefore, we will compare the results to see which method gives the best rotations.

3.4 Multi-Commodity Flow Model

In order to evaluate the generated initial sets of rotations we must solve a Multi-Commodity Flow model (MCF) to flow all containers and compute the corresponding profit or loss. In this application of the MCF the multi-commodity represents the different OD pairs. The model (6)-(11) is implemented and solved in CPLEX. We translate the generated rotations into a directed graph containing several sets of arcs and nodes. The same graph

Table 1: Allowed duration of rotations based on vessel class in rotation generation procedure

Vessel class		Rotation length	Fleet
7500 FFE	(Super Panamax)	[10,10] weeks	0
4200 FFE	(Post Panamax)	[7,14] weeks	0
2400 FFE	(Panamax)	[6,12] weeks	0
1200 FFE	(Panamax)	[4,10] weeks	0
800 FFE	(Feeder)	[1,2] weeks	2
450 FFE	(Feeder)	[1,4] weeks	4

structure is used as the one described in Brouer et al. (2014). Three different sets of nodes exist: source nodes, sink nodes and port nodes. We make a distinction between load (unload) arcs, transshipment arcs , sail arcs and omission arcs.

We start by creating a source node and a sink node for each OD pair. For each pair an omission arc is created between the source and the sink to flow demand that could not be flowed over the rotations. The capacity of this arc equals the total demand of the corresponding OD pair, and the cost equals the revenue per container plus a rejection penalty of 1000 USD. Next, we go over each rotation and add a port node for each port the rotation passes by. Since we are solving a MCF model we need to be able to keep track of which arcs are used by the containers of an OD pair. Therefore, instead of adding one sail arc for every edge in the rotation, we add a sail arc for each OD pair.

It might happen that two rotations pass by the same port, then this port has two nodes. Having a port node per rotation, enables us to create two transshipment arcs between these two port nodes. A transshipment arc has infinite capacity and a cost equal to the transshipment cost per container for that specific port.

Finally, for each port we add load (unload) arcs between all sources (sinks) at that port and all port nodes. Load and unload arcs have infinite capacity and a cost equal to handling cost per container at the port.

In Table 2 an overview is given of all sets, parameters and decision variables used. A small note with respect to the decision variable x_{ij}^k has to be added. As we are flowing thousands of containers it will not be of much relevance whether one container more or less is flowed over a certain arc. Hence, we decided to solve a LP instead of a MIP. This will speed up the solving process of the MCF model.

Table 2: Notation used in MCF model

Sets	
A	Set of all available arcs of the graph
A_l	Set of all load arcs of the graph
A_u	Set of all unload arcs of the graph
A_s	Set of all sail arcs of the graph
A_t	Set of all transshipment arcs of the graph
A_o	Set of all omission arcs of the graph
N	Set of all available nodes of the graph
N_o	Set of all source nodes of the graph
N_d	Set of all sink nodes of the grapgh
N_p	Set of all port nodes of the graph
K	Set of all OD pairs
Parameters	
d_k	Demand for OD pair $k \in K$
r_k	Revenue per container of OD pair $k \in K$
c_{ij}	Cost for using arc $(i,j) \in A$
O_k	Source node for OD pair $k \in K$
D_k	Sink node for OD pair $k \in K$
u_{ij}	Capacity on arc $(i,j) \in A$
Decision var.	
x_{ij}^k	The number of containers of OD pair $k \in K$ to be transported on arc $(i,j) \in A$

$$\max \sum_{k \in K} r_k d_k - \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{6}$$

$$\text{s.t.} \sum_{k \in K} x_{ij}^k \leq u_{ij}, \qquad \forall (i,j) \in A_s \tag{7}$$

$$\sum_{j \in N} x_{ij}^k - \sum_{j \in N} x_{ji}^k = 0, \qquad \forall i \in N_p, k \in K \tag{8}$$

$$\sum_{j \in N} x_{ij}^k = d_k, \qquad \forall i = O_k, k \in K \tag{9}$$

$$\sum_{j \in N} x_{ji}^k = d_k, \qquad \forall i = D_k, k \in K \tag{10}$$

$$x_{ij}^k \geq 0, \qquad \forall (i,j) \in A, k \in K \tag{11}$$

4 Variable Neighborhood Search

4.1 Overview

In Table 3 the resulting objective values for the five different backbone flows are shown. As expected, the quality of the rotations is far from optimal. Therefore, we will apply a network optimization by means of Variable Neighborhood Search (VNS). A VNS is a metaheuristic used for optimization of combinatorial problems. It explores different neighborhoods starting from an initial solution, and only moves away from this solution if an improved solution is found.

This VNS takes as input an initial set of rotations and tries to improve them by exploring different neighborhoods. Subsection 4.2 describes the algorithm used. Subsection 4.3 explains the different neighborhoods.

Table 3: Resulting objective values (in M$) for 20 different sets of initial rotations for each of five different backbone flows

	Best solution value	Median solution value	Worst solution value	Running time (s)
Flow-1	-3.35	-3.44	-4.90	0.0
Flow-2	-3.36	-4.64	-4.90	0.0
Flow-3	-3.35	-4.77	-4.90	0.0
Flow-4	-3.35	-4.90	-4.90	0.0
Flow-5	-3.35	-4.26	-4.90	0.0

4.2 Improvement Algorithm

In Algorithm 2 the pseudocode for the heuristic is given. Each time a local optimum is reached a new starting solution is generated by means of a *shake* and the hill climbing starts over until a new local optimum is reached. A local optimum is reached, when no unused neighborhoods are left. All neighborhoods are set unused if an improved solution is found. As it might happen that the local search finds itself in a loop, only 20 iterations are allowed in one local search. This can happen if it is an improvement to first insert a port and later remove it. We set the stopping criterion for the entire VNS to 30 seconds. Hence, 1 iteration, executing 100 different VNS procedures, will take approximately 50 minutes.

As we are constructing rotations we prefer a relatively modest shake procedure. We want to alter the rotations such that a local optimum is escaped, but we do not want to lose important characteristics of the rotations. In each shake we randomly choose one rotation and alter it by either adding or removing a port call. When adding a port call, we again take into account the maximum travel time and draft of the vessel. In this shake procedure a port call can only be added to a rotation with less than 12 port calls in the

rotation. When removing a port call the minimum number of port calls has to be more than 2. This to maintain feasible rotations.

Algorithm 2 Improvement Algorithm

```
1: Initialization: Find an initial solution x
2: while stopping criterion not met do
3:     generate a new solution x' from x                          ▷ (shake)
4:     while any neighborhood in N is unused do                   ▷ (local search)
5:         choose at random an unused neighborhood and search from x'
6:         if an improved solution x'' is found then
7:             Set x' := x'' and set all neighborhoods unused
8:         if x' is better than x then                            ▷ (move or not)
9:             Set x := x'
10: return x
```

4.3 Neighborhoods

In the VNS we use, 4 different neighborhoods are applied. Each neighborhood seeks to improve the quality of the rotations. As the Baltic fleet is very small and the distances short, we do not seek for the creation of feeder arcs.

4.3.1 Service omission

This neighborhood seeks to improve service at ports with a high amount of omission demand. Omission demand at a port includes demand that left the port on an omission arc, and demand that entered the port on an omission arc. We start by finding the five ports with the highest omission demand. The we randomly choose port $p_{omission}$ to be inserted in one of the rotations.

This neighborhood iterates over each rotation r and finds the closest feasible port $p^r_{closest}$. A port is feasible if $p_{omission}$ can be inserted in an out-and-back fashion such that the maximum length of r and maximum port draft are respected. Then, the altered rotation is chosen with the best evaluated flow.

4.3.2 Service unserved port

This neighborhood seeks to introduce service to unserved ports. An unserved port is randomly chosen for insertion. The probability for a port to be inserted equals its share in the total demand of unserved ports. This means that a port with high demand has a higher chance of insertion than one without. The actual insertion procedure works similar as the one described in the service omission neighborhood.

4.3.3 Remove port

This neighborhood tries to remove a port call from rotation r. It might happen that a port call in rotation r brings more costs than revenue. This is likely when the exchange of cargo is low. The exchange of cargo is defined as the number of loaded plus unloaded plus transshipped containers. It is considered to be low when it is at most 30% of the vessels capacity. If this is the case it might be better to remove this port call. The remove port neighborhood finds for each rotation the best port to be removed. The rotation to be altered is the one with the best evaluated flow.

4.3.4 Simple remove port

This neighborhood seeks to remove port calls which are loading and unloading less than 5% of the vessels capacity. However, to remove a port, at least 2 rotations must call at the port. Otherwise, a port with demand less than 5% of the smallest vessel cannot be served.

5 Allowing for butterfly rotations

The Baltic instance contains 22 demand pairs of which each pair either has Bremerhaven (DEBRV) as its origin or destination. From the data we see that Gothenburg (SEGOT) - which is the second closest port to Bremerhaven - has a weekly outgoing demand to Bremerhaven of 660 FFE and incoming demand from Bremerhaven of 597 FFE. Hence, you could fill a $Feeder_{800}$ for almost 75% of its capacity on the way there and back.

In the current algorithm we only allow for rotations to call at a port at most once. However, from the example given above, it might be profitable to allow multiple calls at a port. Instead of passing by Gothenburg and continue to other ports, it could be better to first return to Bremerhaven and load extra containers for other ports. To further explore the effects of *butterfly* rotations we make a few adjustments in the algorithm. First, in the generation part of the initial rotations we allow for a rotation to call two times at one port. Then, in the VNS we add two more neighborhoods. The neighborhood *add butterfly rotation* and *remove butterfly rotation*.

In the first neighborhood we find for each rotation r which is not yet a butterfly rotation, a port p to add an extra port call to. Port p is a port with the highest demand in r. Then, we determine which arc should be replaced by two return arcs to and from port p. We create a butterfly rotation by replacing arc (i, j) by arcs (i, p) and (p, j). Then, the altered rotation is chosen with the best objective value.

In the second neighborhood we go over each butterfly rotation r and remove the second port call at port p. A butterfly rotation r can also be seen as two sub-rotations r_1 and r_2 connected at p. The second port call is removed as follows. An arc leading to p of

rotation r_1 and an arc leading away from p of rotation r_2 are replaced by one arc combining rotations r_1 and r_2. Hence, we replace arcs (i^r, p) and (p, j^q) with arc (i^r, j^q). Then, the altered rotation is chosen with the best objective value.

6 Optimization of operational costs

In order to evaluate rotations we flowed containers using a MCF model. The model minimized costs for flowing containers over the rotations. These costs solely consisted of cargo-handling, transshipment and omission costs. However, in reality we also have to consider operational costs. Operational costs consist of daily running costs such as crew, repair and maintenance costs. These costs are represented by the TC rate. In addition, we consider bunker and mooring costs.

One might encounter a decision problem. Whether to use less ships on a specific rotation, in order to operate them in another rotation. Then the ships of the first rotation should increase their speed in order to respect the weekly frequency. Or whether to increase speed such that an extra port call can be added to the rotation. However, this brings an increase in bunker costs. Hence, one has to determine whether the increase in revenue of operating another rotation or adding a port call is higher than the costs of extra bunker used.

As mentioned before we are generating and improving rotations at two sailing speeds. In order to not sail faster than needed and waste bunker, we minimize speed such that bunker consumption is minimized while respecting time constraints. To do this, we will solve the model (12)-(18). In Table 4 notation used in speed optimization model can be found.

The problem described above is not one we are likely to face in the small Baltic instance and fleet we have at our possession. However, determining the vessel speed which minimizes bunker consumption while respecting the time constraints, is something we can do. In order to do this, we will solve the model (12)-(18). In Table 4 notation used in speed optimization model can be found.

Table 4: Notation used in speed optimization model

Sets	
R	Set of all rotations in the Baltic
A	Set of all sailing arcs in the graph
Parameters	
x_{ij}^r	Matrix indicating whether $r \in R$ covers $(i,j) \in A$
s_{min,v_r}, s_{max,v_r}	Minimum and maximum speed of vessel type v on r
t_r	Maximum travel time of rotation $r \in R$
d_{ij}	Length of $(i,j) \in A$
Decision var.	
s_r	Speed of vessel on $r \in R$

The objective minimizes the sailing speed. (13) and (14) ensure speed boundaries are respected. Constraint (15) ensures that maximum rotation time is respected. Lastly, (16) defines the domain of the decision variable.

$$\min \quad s_r \tag{12}$$

$$\text{s.t.} \quad s_r \geq s_{min,v_r} \tag{13}$$

$$s_r \leq s_{max,v_r} \tag{14}$$

$$\sum_{(i,j)\in A} \frac{d_{ij}x_{ij}^r}{s_r} + \sum_{(i,j)\in A} p_j x_{ij}^r \leq t_r \tag{15}$$

$$s_r \in \mathbb{R}_{\geq 0} \tag{16}$$

7 Results

In this section we will present all relevant results of both the general and the butterfly algorithm, assuming either design speed of maximum speed . In addition, we show several results obtained by Krogsgaard et al. (2018). For each algorithm 10 replications were executed and data on the best solution was stored. We set the maximum time per VNS procedure at 10% of the maximal running time used by Brouer et al. (2014).

In Table 5 the best, median and average objective values are shown. We see that allowing butterfly rotations, we are able to increase profits at both design and maximum speed. Profits consist of the revenue obtained for flowed containers minus the handling, omission and transshipment costs. We note that both the general as well as the butterfly algorithm were able to outperform the algorithm used by Krogsgaard et al. (2018). The highest profit is obtained when allowing butterfly rotations at maximum speed, namely 1.06 M$. We see that in most cases the average values are smaller than the median values. This might be an indication that the distribution of objective values has a negative skew. In Appendix A elaborate computations are given to demonstrate that the results shown in Table 5 are feasible and correct.

Table 5: Objective value statistics for each algorithm on Baltic instance

Algorithm	Best objective value (M$)	Median Objective value (M$)	Avg. objective value (M$)	Repli-cations	Running time (s)
General (design)	0.65	-1.16	-1.21	10	30
General (max)	0.78	-1.20	-1.62	10	30
Butterfly (design)	0.72	-0.61	-0.93	10	30
Butterfly (max)	1.06	-1.08	-1.24	10	30
Krogsgaard	0.09	-0.14	-0.15	10	30

In Table 6 some statistical data is shown regarding the algorithmic performance of the algorithms. The table shows that both the general and butterfly algorithm are must faster in evaluating neighborhoods. The general algorithm performs almost six times more iterations. An iteration is considered one local search, so a search until all neighborhoods are used or the number of loops reached its limit. Also, each algorithm has more improving iterations. This could explain the fact a better objective value is reached by the first four algorithms. If a VNS runs faster, more local optima can be explored and hence a higher probability of reaching a higher optimal objective.

Table 6: Algorithmic performance for the best solution of each algorith on Baltic instance

Algorithm	Iterations	Improving iterations	Last improving iteration	Avg. time per iteration (s)
General (design)	713	17	439	0.04
General (max)	517	16	92	0.06
Butterfly (design)	409	18	99	0.07
Butterfly (max)	249	9	150	0.12
Krogsgaard	85	6	50	0.4

Table 7 shows percentages of improving iterations caused by each neighborhood. We see that for each algorithm most improving iterations were caused by the service-omission neighborhood. This is an indication that the initial rotations contained a relatively high omission flow. Another striking observation is the fact the simple remove port neighborhood never resulted in an improved solution. Furthermore, it can be seen that removing a butterfly rotation was never beneficial for the objective value.

Table 7: Percentages of improving iterations caused by each neighborhood

Algorithm	Omission	Unserved	Remove	Simple Remove	Add b.fly	Remove b.fly
General (design)	59%	18%	23%	0.0%	NA	NA
General (max)	69%	19%	12%	0.0%	NA	NA
Butterfly (design)	39%	33%	11%	0.0%	17%	0.0%
Butterfly (max)	44%	0.0%	33%	0.0%	23%	0.0%

Figure 3 demonstrates the progress of the optimal objective value versus the current value over time. We see that VNS reaches the optimal solution in several steps. After each increasing step the current value cycles for a while until a move terminates the cycle.

Table 8 gives some information regarding the number of rotations, average length per rotation, average number of port calls per rotation and percentages of rejected cargo.

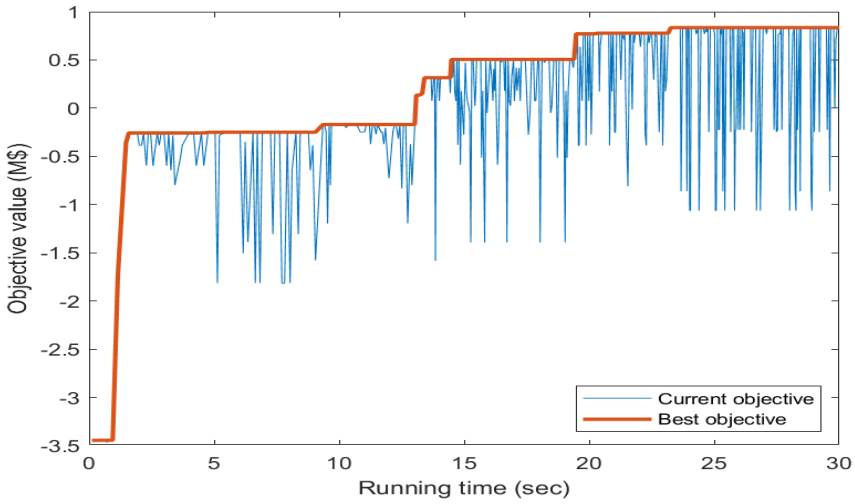

Figure 3: Development of objective value in the Baltic instance

The butterfly algorithms visit more ports than the general algorithms. We see that the butterfly algorithm with maximum speed has the same number of rotations as Krogsgaard, however manages to visit two ports more on average per rotation. We note that as speed increases, the number of ports visited increased and hence, the rejected cargo decreases.

Table 8: Properties describing the rotations of the best solution for each algorithm

Algorithm	Deployed capacity	Rotations	Avg. rot. length (1000nm)	Avg. port calls per rot.	Rejected cargo
General (design)	100%	4	1.91	3.75	19.62%
General (max)	100%	4	2.10	4.50	17.29%
Butterfly (design)	100%	3	2.50	5.67	18.58%
Butterfly (max)	100%	2	4.08	9.00	12.29%
Krogsgaard	100%	2	4.55	7.00	8.00%

Then, in Table 9 the construction of the total operational costs is given of the general algorithm assuming design speed. For the tables of the remaining three algorithms Appendix B can be consulted. As mentioned before, these costs consist of bunker, operational and port call costs. The bunker costs are computed as follows. First, the bunker consumption per day at minimized speed is computed using the following cubic function presented in Brouer et al. (2014):

$$F(s) = (s/v_*^F)^3 f_*^F \tag{17}$$

for any minimized speed s between the *min speed* s_{min}^F and *max speed* s_{max}^F of the vessel F , where v_*^F is the design speed, and f_*^F is the fuel consumption at design speed. Then, we can compute the amount of bunker used and hence, the cost for all bunker used. In addition, we consider the operational costs which are given by the TC rate per day. We are assuming that these rotations are sailed with infinite time horizon, hence there are always

two vessels operating. Thus, the operational cost is given by $\#_{days} * \#_{vessels} * TCrate$. Lastly, we consider the costs for calling at ports. These consist of variable cost relative to the capacity of the vessel calling at the port, and a fixed port call cost.

Table 9: Total revenue of sailing rotations generated by general algorithm with design speed (costs and profit in M$)

Number vessels	Minimized speed	Distance sailed	Bunker used	Bunker costs	tcRate	days	Oper. costs	portcall costs
1	12,09	1161	69	0.04	8000	7	0.06	0.05
1	12,04	1156	68	0.04	8000	7	0.06	0.17
2	11,32	2446	154	0.09	5000	14	0.14	0.25
2	12,00	2864	197	0.12	5000	14	0.14	0.14
Total cost	1.30							
Profit	-0.65							

We see that highest profit is obtained for the general algorithm with design speed, namely -0.65 M$. This value is obtained after subtracting the total cost in the tables above, from the objective value given in Table 5. However, this is the profit when sailing all rotations once over a time spawn of two weeks. If you consider this set of rotations as one starting every two weeks, then this set can be sailed 18 times in 36 weeks. In this manner we can convert the profits such that they become comparable. Table 10 exhibits these converted profits. We note that the highest profit is obtained for butterfly rotations which were generated assuming maximum speed, namely -7.9 M$. Thus, no algorithm managed to obtain profitable rotations. The best set of rotations is shown in Figure 4. The rotations of the remaining three algorithms can be found in the Appendix C.

Table 10: Revenue minus operational costs per 36 weeks

Algorithm	General (design)	General (max)	Butterfly (desing)	Butterly (max)
Profit	-11.7	-17.5	-14.4	-7.9

8 Conclusion

The goal of this paper was to construct a cost and energy efficient solution to the LSNDP. A two-stage approach was used in which first containers were flowed over a relaxed network in order to create a backbone flow. Then, a greedy construction heuristic was applied to sequentially create rotations based on the backbone flow. A distinction can be made between four different algorithms. In the construction heuristic two different versions

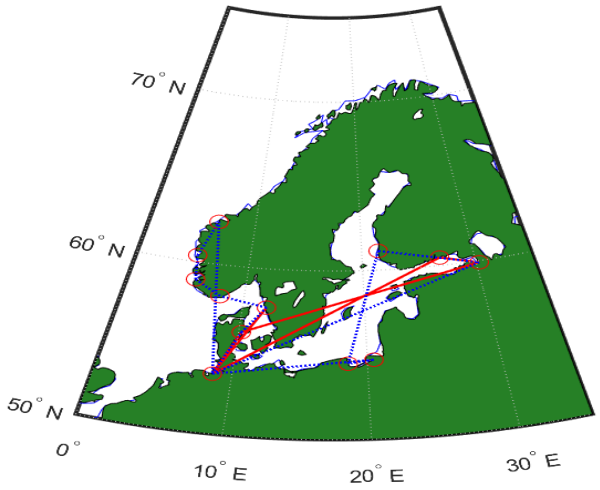

Figure 4: Best found rotations for Baltic instance using butterfly algorithm with maximum speed

of rotations were created, one in which a rotation can only call once at each port, also referred to as general, and one in which butterfly rotations are allowed. For both versions two sets of rotations were constructed. In one set we assumed that the vessels sailed at design speed and the other that they sailed at maximum speed. When sailing at maximum speed, more ports can be visited on a single rotation, however higher operational costs are the consequence. We created 5 backbone flows and for each flow 20 sets of rotations for each algorithm. These were used as initial solutions for the improvement algorithm.

After having generated the initial solutions we applied a VNS algorithm which explored 4 and 6 different neighborhoods, for the general rotations and butterfly rotations respectively. Then, as a last step, we minimized the sailing speed of each rotation such that the rotation remained feasible, while minimizing operational costs and bunker usage.

We managed to increase the revenue obtained for flowing containers significantly. Highest increase was observed for the rotations which are sailed at maximum speed, which was to be as expected as more ports can be visited. However, these rotations are also more costly. After subtracting operational costs from the revenue and comparing the profits obtained over 36 weeks, we got as result that no profitable solutions were found. We conclude the butterfly algorithm assuming maximum speed to be the best algorithm as it proved to be the least loss-making.

For future research it might be of value to implement time constraints on the transported cargo. Also, an important factor not considered in this paper is the repositioning of containers. Dong and Song (2009) found that 27% of container traffic is empty repositioning. Therefore, incorporating the repositioning of containers in a LSNDP could result in better and more accurate results.

A Results explained

In order to show that the results obtained are valid and feasible, we demonstrate for one set of rotations created that all requirements are met. We use the rotations generated by the butterfly algorithm using maximum speed.

The butterfly algorithm using maximum speed finds in the best solution two butterfly rotations, one for $Feeder_{800}$ with a maximum rotation time of 2 weeks, and one for $Feeder_{450}$ with a maximum rotation time of 4 weeks. The rotations are as follows:

- $Feeder_{800}$: $\{0 \to 11 \to 0 \to 2 \to 10 \to 1 \to 0\}$
- $Feeder_{450}$: $\{0 \to 9 \to 8 \to 3 \to 10 \to 0 \to 1 \to 11 \to 6 \to 7 \to 5 \to 4 \to 0\}$

From our algorithm we obtained that the only demand flowed over an omission arc was of OD pair 20, namely 603 containers. In Table 11 all OD pairs are given with their corresponding weekly demand. In Table A we exhibit the number of containers of each OD pair flowed over each arc. We can see that on each arc the capacity is not surpassed. Hence, all containers, except for the omission flow of OD pair 20, are flowed in a feasible manner.

Table 11: All OD pairs with corresponding weekly demand

ID	Origin	Destination	Demand	ID	Origin	Destination	Demand
1	FIRAU(3)	DEBRV(0)	77	12	NOBGO(5)	DEBRV(0)	37
2	DEBRV(0)	DKAAR(1)	456	13	DEBRV(0)	FIKTK(2)	187
3	DEBRV(0)	NOSVG(7)	65	14	NOAES(4)	DEBRV(0)	50
4	RUKGD(9)	DEBRV(0)	7	15	PLGDY(8)	DEBRV(0)	231
5	DEBRV(0)	NOAES(4)	10	16	DEBRV(0)	SEGOT(11)	597
6	DEBRV(0)	PLGDY(8)	98	17	NOSVG(7)	DEBRV(0)	32
7	SEGOT(11)	DEBRV(0)	660	18	FIKTK(2)	DEBRV(0)	162
8	DEBRV(0)	NOBGO(5)	17	19	DKAAR(1)	DEBRV(0)	397
9	DEBRV(0)	RUKGD(10)	268	20	DEBRV(0)	RULED(10)	1215
10	DEBRV(0)	FIRAU(3)	18	21	DEBRV(0)	NOKRS(7)	6
11	NOKRS(6)	DEBRV(0)	16	22	RULED(10)	DEBRV(0)	298

The total revenue that can be obtained equals \$4054660, which is given by multiplying the demand by the revenue for each OD pair. Costs for flowing the containers consist of handling, transshipment and omission costs. We handle all containers except for those that flow on omission arcs. Hence, we get a handling costs of \$2030566 and an omission cost which equal \$958770. This leaves us with a revenue of \$1065324, which is still a bit more than the rounded best objective value given in Table 5, which unrounded equals \$1061860. This can be explained by the fact several containers are transshipped from one rotation to the other at ports 10 and 11.

Table 12: Number of containers of each OD pair on each arc (OD pair in brackets)

Arc	Number of containers (OD pair)	Load	Capacity
$(0, 11)$	10(3),10(5),597(16),6(21)	623	800
$(11, 0)$	650(7)	650	800
$(0, 2)$	68(2),187(13),545(20)	800	800
$(2, 10)$	68(2),162(18),545(20)	775	800
$(10, 1)$	68(2),162(18),163(22)	393	800
$(1, 0)$	162(18),397(19),163(22)	722	800
$(0, 9)$	98(6),268(9),18(10),66(20)	450	450
$(9, 8)$	7(4),98(6),28(10),66(20)	199	450
$(8, 3)$	7(4),28(10),231(15),66(20)	332	450
$(3, 10)$	77(1),7(4),231(15),66(20)	381	450
$(10, 0)$	77(1),7(4),231(15),135(22)	450	450
$(0, 1)$	388(2),54(3),7(8)	449	450
$(1, 11)$	54(3),7(8)	61	450
$(11, 6)$	54(3),10(5),10(7),17(8),6(21)	97	450
$(6, 7)$	54(3),10(5),10(7),17(8),16(11)	107	450
$(7, 5)$	10(5),10(7),17(8),16(11),32(17)	85	450
$(5, 4)$	10(5),10(7),16(11),37(12),32(17)	105	450
$(4, 0)$	10(7),16(11),37(12),50(14),32(17)	145	450

B Tables operational costs

Table 13: Total revenue of sailing rotations generated by general algorithm with maximum speed (costs and profit in M$)

Number vessels	Minimized speed	Distance sailed	Bunker used	Bunker costs	tcRate	days	Oper. costs	portcall costs
1	16.82	1615	172	0.10	8000	7	0.06	0.17
1	16.47	1186	126	0.08	8000	7	0.06	0.18
2	13.61	2614	234	0.14	5000	14	0.14	0.26
2	13.90	3002	275	0.17	5000	14	0.14	0.25
Total cost	1.75							
Profit	-0.97							

Table 14: Total revenue of sailing rotations generated by butterfly algorithm with design speed (costs and profit in M$)

Number vessels	Minimized speed	Distance sailed	Bunker used	Bunker costs	tcRate	days	Oper. costs	portcall costs
2	13,78	2616	194	0.12	8000	14	0.22	0.25
2	11,77	1641	123	0.07	5000	14	0.14	0.33
2	11,84	2318	161	0.10	5000	14	0.14	0.15
Total cost	1.52							
Profit	-0.80							

Table 15: Total revenue of sailing rotations generated by butterfly algorithm with maximum speed (costs and profit in M$)

Number vessels	Minimized speed	Distance sailed	Bunker used	Bunker costs	tcRate	days	Oper. costs	portcall costs
2	16,55	3178	328	0.20	8000	14	0.22	0.17
4	12,99	4989	410	0.25	5000	28	0.56	0.54
Total cost	1.94							
Profit	-0.88							

C Resulting rotations

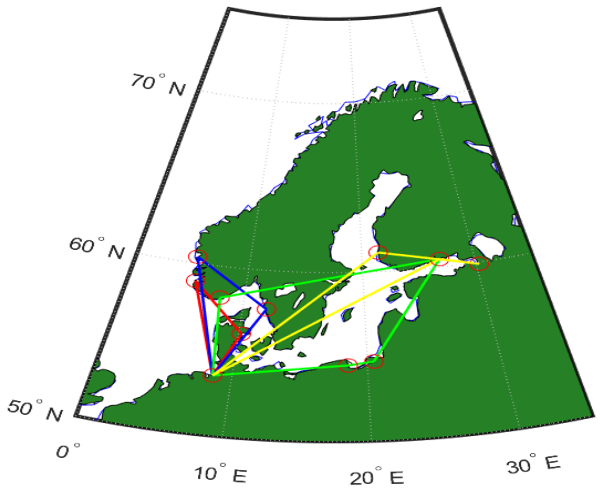

Figure 5: Best found rotations for Baltic instance using general algorithm with design speed

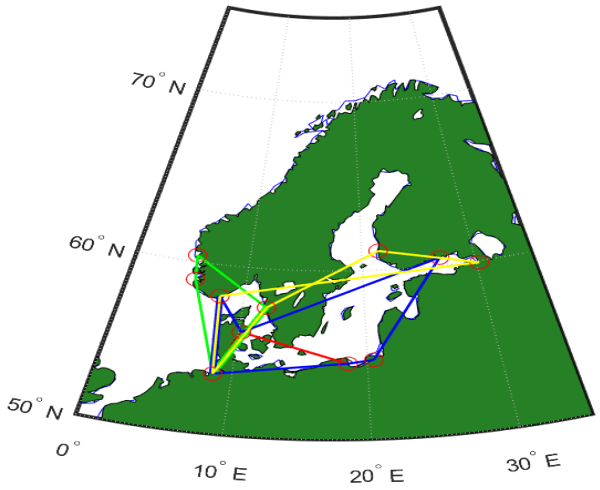

Figure 6: Best found rotations for Baltic instance using general algorithm with maximum speed

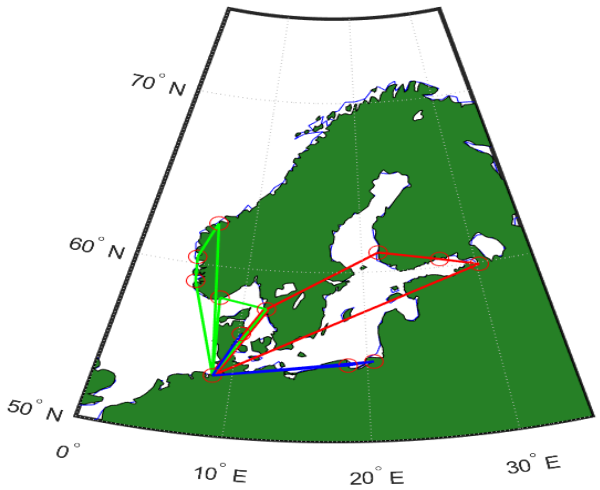

Figure 7: Best found rotations for Baltic instance using butterfly algorithm with design speed

D Java files described

A distinction can be made between four different packages: *ReadAndCompute*, *Backbone-Flow*, *General* and *Butterfly*.

Package *ReadAndCompute:*
This package consists of several java files which were used to read the big data files of LINER-LIB into more compact text files which are used in the other packages as input.

Package *BackboneFlow:*

- Model: object that solves the model (1)-(5) from Section 3.2.
- Main: this class creates the backbone flow by repeatedly creating and solving a Model. It prints the backbone flows to text files.

Package *General:*
This package contains all java files necessary for the general algorithm for both the design speed as well as the maximum speed. Only one line of code has to be changed when changing the speed, namely exchange line 37 with line 39 in *GenerateRotations.java*. This package thus generates the initial rotations and runs the VNS. In case you like to run the general algorithm one only has to run *Main.java*.

- Node: object containing info for a port node, such as which port it is and to which rotation it belongs to.
- Edge: object containing info for an edge, such as the type of edge and to which OD pair the edge belongs.
- ODpair: object containing info for an OD pair, such as origin, destination and demand.
- RotationClass: object containing important info regarding a rotation, such as the nodes in the rotation, predecessor and successor of each node and the vessel class sailing the rotation.
- GraphClass: object containing a graph $G = (V, A)$ together with all important info regarding the graph itself.
- AlgPerformance: object containing info regarding the algorithmic performance of VNS.
- RandomCollection: object used to return weighted random objects.
- GenerateRotations: object that generates all sets of initial rotations.
- ModelMCF: object that solves the MCF in (6)-(11) in Section 3.4.
- VNS: object that executes the VNS described in Section 4.
- ModelSpeed: object that minimizes sailing speed by solving model (12)-(16) in Section 6.
- Main: this class creates all objects in the right order and executes the entire algorithm. Results which are presented in Section 7 were printed out to text files in the Main.

Package *Butterfly:*
This package contains the exact same java files, however two neighborhoods are added to the code in VNS.java and some minor adjustments in other java files. Executing the butterfly algorithm is done in the same manner as described above in the General Package.

References

[1] Alvarez, J.F. (2009). *Joint routing and deployment of a fleet of container vessels.* Maritime Econom. Logist. 11(2):186–208.

[2] Brouer, B.D., & Fernando Alvarez, J., & Plum, C.E.M., & Pisinger, D., & Sigurd, M.M. (2014). *A Base Integer Programming Model and Benchmark Suite for Liner-Shipping Network Design.* Transportation Science 48(2):281-312.

[3] Brouer, B.D., & Desaulniers, G., & Pisinger, D. (2014). *A matheuristic for the liner shipping network design problem.* Transport.Res.ELogist.Transport.Rev. 72 (2014), 42–59.

[4] Dong, J-X, & Song, D. (2009). *Container fleet sizing and empty repositioning in liner shipping systems.* Transportation Res. Part E: Logist. Transportation Rev. 45(6):860–877.

[5] Hansen, P., & Mladenovic, N. (2014). Variable neighborhood search. In E.K. Burke & G. Kendall (Eds.), *Search Methodologies - Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 313-337). New York: Springer.

[6] Krogsgaard, A., & Pisinger, D., & Thorsen, J. (2018). *A flow-first route-next heuristic for liner shipping network design.* Networks 72, 358-381. https://doi.org/10.1002/net.21819

[7] Pisinger, D.(2016). *Liner shipping network design–a new decomposition,* Conference Handbook EURO-2016, 3–7 July, Poznan, Poland.

[8] Stopford, M.(2009). *Maritime Economics,* 3rd ed. (Routledge, Oxford, UK).

[9] UNCTAD (2016). *Review of maritime transport.* Technical report, available at unctad.org/en/PublicationsLibrary/rmt2016en.pdf.