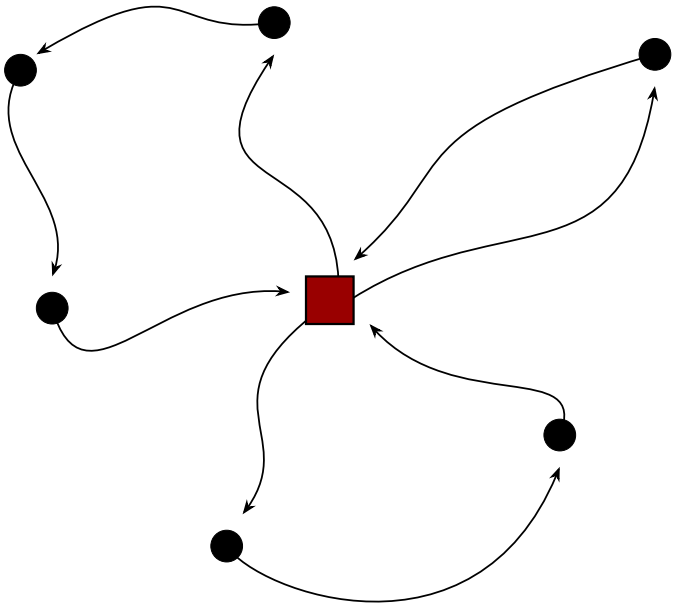
TECHNICAL UNIVERSITY OF DENMARK
BSc. in Data Science & Management

Optimal Maritime Connectivity

Algorithmic Modeling of Liner Shipping Networks

Johan Matzen Brodt (s204607) & August Høgsted (s204630)
Supervised by Stefan Røpke



December 22, 2023

Abstract

This Bachelor thesis explores the application of a two-stage algorithm to address the challenges posed by the Liner Shipping Network Design problem (LSNDP). The LSNDP involves creating and optimizing a network for liner shipping routes, presenting a complex optimization challenge. To tackle this complexity, advanced algorithms are required to efficiently optimize the problem without considering its entirety. The two-stage algorithm is a widely employed method and has demonstrated promising results in solving the LSNDP. Specifically, the service-first flow-second algorithm is considered in which services are constructed, whereafter, the cargo-flow problem is solved to minimize overall costs. The proposed method is tested on three out of the seven base instances of the LINER-LIB benchmark suite, namely the Baltic, WAF, and Mediterranean.

In the service-first stage, the simulated annealing (SA) metaheuristic generates and optimizes services, focusing on minimizing fixed costs. The SA metaheuristic includes two hyperparameters, which are exposed to a grid search to optimize the performance of the metaheuristic. Once the optimal hyperparameters are determined, a set of services is generated, which functions as the foundational network of the flow-second stage. All services offer transshipment options between them, ensuring that commodities has a feasible route from its origin port to its destination port.

The flow-second stage is constructed as a path-based Mixed-Integer Program (MIP) whose structure enables the utilization of Dantzig-Wolfe decomposition using column generation to overcome the complexity of the model. This involves relaxing binary variables to retrieve the dual variables of the restricted master problem (RMP), allowing for calculating the reduced cost of the considered paths. The basis of the optimal RMP is utilized in the MIP, where the binary variables are tightened to obtain a valid solution to the LSNDP.

The results of the two-stage algorithm indicate that the SA metaheuristic's network design does not perform consistently well across the three instances, emphasizing the need for a more tailored approach. The MIP encountered challenges with lower delivery rates in the Baltic and Mediterranean instances, which might be attributed to smaller fleets. This underscores the need to refine the SA metaheuristic's logic for constructing services. Conversely, WAF showed promising results with the given algorithm for network design, yielding profits and high delivery rates.

Keywords: LINER-LIB, Liner Shipping Network Design Problem, Simulated Annealing metaheuristic, Dantzig-Wolfe decomposition, Column generation, Service design, MIP, Optimization.

Nomenclature

Berth: Specific location at a port where a vessel (un)loads cargo

FFE: Defined as Forty-Foot-Equivalent-Unit and is the size for a 40ft standard shipping container

Hub: Ports with more than 20 market orders are defined as hubs

Idle: When a vessel berths at a port, it idles for 24 hours before it can sail to the next port

IP: Integer Program

LSNDP: Liner Shipping Network Design Problem

Market order: Term from the stock world that means to buy or sell a stock. In this thesis, it refers to all demand- and supply-calls of a port

LS-MCFP: Liner Shipping Multi-Commodity Flow Problem

MCFP: Multi-Commodity Flow Problem

MIP: Mixed-Integer Program

MST: Minimum Spanning Tree

Port Calls: When a vessel berths at a port

RMP: Restricted Master Problem

SA: Simulated Annealing

Service: Cyclic itineraries that vessels follow

SND: Service Network Design

Contents

1 Introduction

Global trade, the backbone of the modern world's interconnected economies, relies heavily on the maritime industry, particularly liner shipping. Liner shipping plays an incremental role in ensuring the movement of goods across the world's oceans, facilitating efficient trade among manufacturers, suppliers, and consumers. Maritime shipping is by far the most widespread transportation mode of goods, accounting for over 90% of all cargo transportation in the world (Economic Co-operation and Development n.d.). The largest shipping companies possess the capability to transport millions of *forty-foot equivalent unit* (FFE) using fleets consisting of hundreds of ships (Maersk n.d.). Needless to say, small reductions in cost would have a tremendous impact on the profit made.

The expansive nature of the liner shipping industry presents challenges that need addressing to establish an efficient and profitable supply chain. These challenges encompass transit times of demands, lowering emissions, reducing costs, constantly changing schedules, and consistently fulfilling customer requirements. One can optimize its operation by introducing liner shipping routes, also called services, to prevent these challenges from triggering significant disruptions leading to financial setbacks. Services are defined as cyclic itineraries that vessels follow when transporting cargo. A liner shipping network refers to a predefined and consistent set of services on which vessels operate. This approach ensures that customers benefit from consistent and reliable cargo transportation, reducing supply chain uncertainties.

This thesis aims to solve the LSNDP by establishing a liner shipping network constituting services while utilizing vessel performance in terms of capacity, traversable distance, and cost. The services will be tested on three instances of the LINER-LIB benchmark suite. These instances comprise a particular sea consisting of ports, including specified demands (commodities) and fleets. The methodology involves a two-stage algorithm called service-first flow-second (Christensen et al. 2021). The initial stage is dedicated to the network design, where a set of services S are generated using a metaheuristic. The second stage involves optimizing the cargo flow of each instance using a MIP. The problem's structure presents an opportunity to leverage the Dantzig-Wolfe decomposition method, effectively minimizing the computational demands of solving the models. The findings will be compared to alternative methodologies and approaches, particularly concerning the two-stage algorithm.

The structure of this thesis is as follows: Section 2 reviews current literature related to the two-stage algorithm and the LINER-LIB benchmark suite. Section 3 describes the LSNDP and provides an overview of the variables involved in the modeling. Section 4 explains how the service-first stage constructs services through the SA metaheuristic

and presents the logic behind its implementation. Section 5 explains the network modifications to model shipping activities properly, including (un)loading cargo, forfeiting cargo, and transshipment of cargo. Section 6 presents the mathematical formulation of the flow-second stage and how column generation can be utilized to construct feasible paths that will finally be used to solve the MIP. Section 7 analyses the experimental results, including hyperparameter optimization for the SA metaheuristic and the results from solving the MIP. Section 8 discusses critical choices and obtained results. The thesis' core findings are summarized in section 9. Interesting topics for future work are described in section 10, and a retrospective assessment of the initial project plan and learning objectives are found in section 11. Lastly, the appendices can be seen in the appendix. A foundational understanding of operations research and network optimization is recommended to grasp the concepts presented in this thesis.

2 Literature review

The literature on the LNSDP has seen significant growth and diversification in the past decade. This problem is an effective instrument for modeling shipping operations' intricate and dynamic facets, encompassing route optimization, transit time constraints, and speed optimization within maritime transportation. Researchers and practitioners have delved into various methods to tackle the LSNDP, primarily using two-stage algorithms, which have shown great success so far (Christensen et al. 2021).

The Álvarez (2009) paper presents one of the early two-stage approaches that consider the joint routing and fleet deployment problem. The problem is formulated as a MIP model that minimizes the operating expenses of a liner company over a tactical planning horizon. The paper suggests a smart exploitation of MIP by transforming the problem into a Multi-Commodity Flow problem (MCFP). This approach enables solving large instances by predefining the number of vessels assigned to each combination of vessel type, speed, and service. A tabu search is introduced to determine the optimal values of the predefined decision variables. The tabu search utilizes the solution from the MCFP to improve the vessel deployment iteratively. The tabu search is solved using column generation to mitigate the computational demand, retrieving the dual solution from the MCFP of the previous iteration. This enables the generation of numerous combinations of vessel deployments.

Brouer et al. (2014) expands upon the mathematical formulation introduced in Álvarez (2009) and takes butterfly rotations and (bi)weekly service frequencies into account. Butterfly rotations enable a single port, the butterfly port, to be visited several times. An illustration of the butterfly service is found in figure 2 in section 4.

In addition, Brouer et al. (2014) provides a benchmark suite, LINER-LIB, for solving the LSNDP to enable access to the liner shipping domain and the data sources of liner shipping. The LINER-LIB data set facilitates higher-quality assessments and comparisons among various algorithmic approaches for solving the LSNDP. The benchmark suite is used to formulate a more advanced version of the MIP presented in Álvarez (2009). The MIP is solved as a MCFP in which the objective is to minimize the daily running costs incurred by vessels in operation and the costs obtained when excluding vessels from the operating fleet. A tabu search is then applied to find the most promising set of rotations generated by an underlying auxiliary problem of the MIP. While their advanced LSNDP formulation produces high-quality rotations, future modeling work must be considered incorporating commercial requirements, such as low transit times and slow steaming, into current network design models.

The paper by Karsten et al. (2017) incorporates time constraints into the LSNDP, allowing for an arbitrary number of transshipments and enabling restrictions on the transit time of commodities. Transit time refers to the duration taken to transport a container from its origin to its destination. The approach consists of a heuristic, namely SA, that improves the quality of the services by searching in an extensive neighborhood space. Here, the heuristic constructs and improves the services by inserting or removing port calls. In addition, amendments are made to the fleet deployment along with a proposed sailing speed on each service leg, leading to potential improvements in the overall profit. The improvement of the services is reflected after resolving a MCFP using column generation in each iteration. Further, an Integer Program (IP) is constructed as a path formulated LSNDP in which transit time restrictions are considered in the definition of a feasible path for a given commodity. The IP aims to minimize both service and commodity transportation costs combined. The method is evaluated using LINER-LIB as described in Brouer et al. (2014). Results showcase the approach's ability to create highly profitable networks, particularly in larger LINER-LIB instances. In addition, the paper suggests that optimizing speeds on individual legs and allowing vessel class swaps could reduce rejected cargo and enhance fleet utilization.

Koza, Desaulniers, and Røpke (2020) present an integrated LSNDP and scheduling problem, meaning that transshipment times depend on the exact itinerary of the vessels within the services. The proposed method incorporates advanced shipping elements, such as leg-based speed optimization, without being tied to specific service rotations such as simple or butterfly services. Koza, Desaulniers, and Røpke (2020) provide a MIP formulation for the LSNDP, which aims to minimize the total cost of the services and the container flows. The MIP employs a column generation matheuristic, functioning

as an iterative destroy-and-rebuild mechanism to reoptimize the network. The column generation aims to determine the optimal commodity paths from a set of feasible solutions. In addition, a polishing heuristic is applied to diversify and enlarge the set of feasible scheduled networks. The approach's complexity aligns with its remarkable outcomes, showcasing superior solutions across all instances of the LINER-LIB benchmark suite. One notable drawback to this solution approach is that it is computationally heavy. Furthermore, the findings suggest that disregarding scheduling and approximating transshipments rather than calculating them accurately might result in liner shipping networks underestimating commodity transit times and their implications.

The LINER-LIB benchmark suite is versatile, accommodating diverse approaches distinguished by unique combinations of algorithms. Given the proven success of the two-stage approach in prior studies, this thesis further investigates the service-first and flow-second approach. Due to the computational demand and limited time, the feedback mechanism of the mathematical model will not be considered. Hence, this paper can be interpreted as a simple implementation method for solving the LSNDP, offering a straightforward approach.

3 Problem description and data

3.1 Liner shipping network design problem

The LSNDP aims to optimize liner shipping routes by minimizing the total cost of the network, including cargo flow. The LSNDP is complex and challenging to solve, as there are many rules and constraints that must be adhered to. The problem can be defined as follows: Given a collection of ports, a fleet of container vessels, and a group of demands, a set of services $\in S$ is constructed for the container vessels such that the overall operational expenses are minimized, while ensuring that all demands can be routed through the resulting network, respecting the capacity of the vessels (Christensen et al. 2021). The demands correspond to the volume of FFE required to be transported from an origin to a destination and are referred to as commodities. These commodities are accompanied by precise details outlining the demand in FFE and the revenue generated per FFE. The revenue signifies the monetary gain of delivering one FFE to the corresponding destination port. The LSNDP is depicted in figure 1.

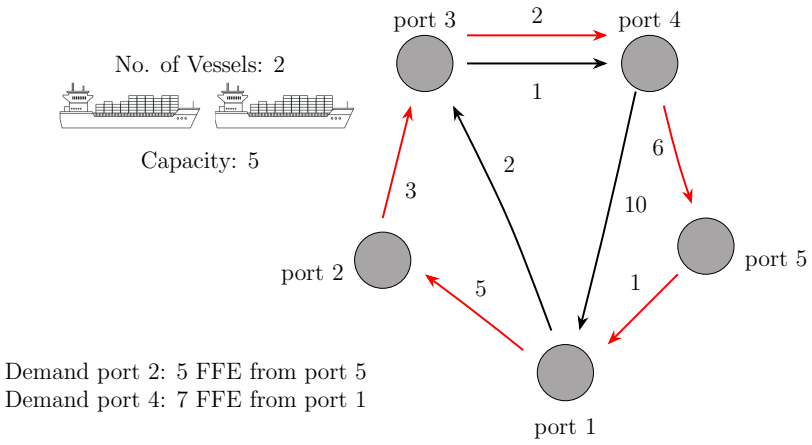


Figure 1: Visual representation of the LSNDP. Five ports are shown where two ports have a demand, and two ports are the origins of those demands. The numbers on the edges between them denote the cost of operation from one port to another. The edges show which paths a feasible to traverse. The figure shows two arbitrary services, each denoted by the different colors. Two vessels, each with a fixed cost, are available for deployment on the services. The idea behind the LSNDP is to construct services that maximize the flow of containers to accommodate the demand while minimizing operating costs.

3.2 LINER-LIB benchmark suite

This thesis is built upon the LINER-LIB, a benchmark suite for solving the LSNDP. The benchmark suite comprises seven test instances derived from real-life data from Maersk Line, supplemented by data from several industries and public stakeholders. This benchmark facilitates developing and implementing industry-relevant models and methods concerning liner shipping network design- and fleet deployment problems at a strategic level (Brouer et al. 2014). Each instance contains specified commodity- and fleet data. The benchmark suite enables modeling three distinct scenarios related to different capacity/demand relations from each of the seven instances. This thesis only touches on the base scenario of the Baltic, WAF, and Mediterranean instances. Table 1 shows an overview of the three instances. Baltic is the smallest of the three instances, with 12 ports and 22 commodities; WAF has 20 ports and 37 commodities; and the Mediterranean is the largest, with 39 ports and 365 commodities. The ports within the instances are referred to by their UN/LOCODE. The available fleets of each instance can be found in their respective fleet data containing information about vessel classes and quantity of the vessels.

Instance	Category	$\|N\|$	$\|K\|$	V	Total c^F	Total q^C
Baltic	Single-hub	12	22	4 x class 1 2 x class 2	3,400	4,904
West Africa	Single-hub	20	37	14 x class 1 28 x class 2	28,700	8,541
Mediterranean	Multi-hub	39	365	8 x class 1 8 x class 2 4 x class 3	14,800	7,545

Table 1: Overview of the instances from LINER-LIB. $|N|$ is the number of ports, $|K|$ is the number of commodities, total c^F is the capacity of all vessels, and total q^C is the quantity of all commodities. The first vessel class is Feeder_450, the second is Feeder_800, and the last vessel class is Panamax_1200.

3.3 Attributes

To ensure alignment for convenient comparisons with existing and future literature, this thesis uses the same attribute names and notations from the original paper titled *A Base Integer Programming Model and Benchmark Suite for Liner-Shipping Network Design* (Brouer et al. 2014). The following attributes are considered in solving the LSNDP.

The cargo demand list shown in table 2 contains the *origin*- and *destination* port of all commodities in a given instance, and the *quantity* is the size of the order in FFE. The *freight rate* is the revenue per FFE delivered in USD.

The attributes of the ports of each instance can be seen in table 5. This table includes information about the *port*, given as a UN/LOCODE, which is an international coding scheme (comission n.d.). *Name* is the name of the port, and *country* is the origin country of the port. *Latitude* and *Longitude* represents the geographic coordinates of the port. *Move cost* is the cost of (un)loading cargo from a vessel in USD per FFE. *Transhipment cost* is the cost of performing a transshipment in a port. Lastly, *Fixed port call cost* is a fixed price paid each time a vessel berths at a port.

The distance between ports is given in the distance list shown in table 3. It is an all-to-all distance list with the *distance* measured in nautical miles between the *origin* and the *destination* ports.

The fleet list in table 4 contains information about the instances' different *vessel classes*. The *capacity* is the amount of FFE a vessel can carry. The *design speed* is the speed at which a vessel sails in nautical miles per hour. The *quantity of vessel* is the number of vessels belonging to a unique vessel class in each instance. The *TC rate* is

the cost per day of having a vessel employed in a service in USD. The *fuel consumption, design speed* is the total consumption of bunker fuel per day in tons when sailing at design speed s_*^F. Lastly, the *fuel consumption, idle* is the total consumption of bunker fuel per day in tons when the vessel is idling.

Cargo demand list C
Origin port α^C
Destination port β^C
Quantity q^C
Freight rate f^C

Table 2: Cargo demand attributes

Distance list D
Origin port α^D
Destination port β^D
Distance d^D

Table 3: Distance attributes

Fleet list V
Vessel class v^F
Capacity c^F
TC rate T^F
Design speed $s_\star^F$
Fuel consumption $f_\star^F$, design speed
Fuel consumption f_0^F, idle
Quantity of vessel q^F

Table 4: Fleet attributes

Port list P
Port α^P
Name n^P
Country o^P
Latitude x^P
Longitude y^P
Move cost m^P
Transshipment cost t^P
Fixed port call cost f^P

Table 5: Port attributes

4 Service design

Service Network Design (SND) encapsulates the intricate decisions and complexities inherent in planning the supply of cargo within a transportation system. The planning must satisfy a given or estimated demand while being efficient, profitable, and compliant with industry standards.

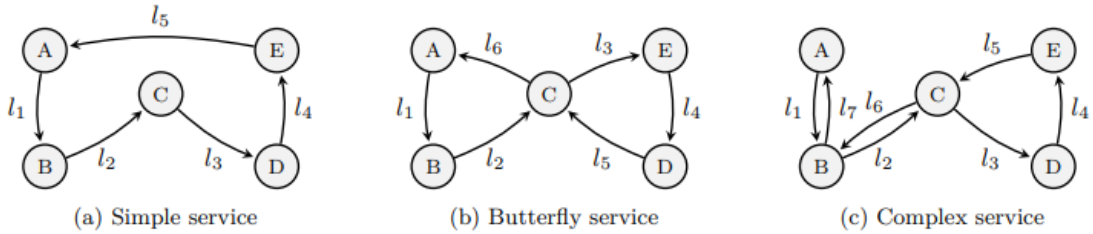
The overarching trade-off of the SND problem lies between leveraging economies of scale and meeting the required demand from customers for faster and more cost-effective transportation. This dilemma often plays out in scenarios where enhancing customer satisfaction entails increasing service frequency within a given period, which, on the other hand, limits resources for other services and increases terminal congestion (Teodor and Hewitt 2021). Another crucial challenge that must be addressed concurrently involves constructing a service that provides optimal connectivity between ports. The intricate nature of configuring services leads to a vast complexity characterized by an expansive solution space comprised of numerous combinatorial configurations. To

discover this solution space effectively, a SA metaheuristic is applied to construct the best possible services for each considered instance of the LINER-LIB benchmark suite. The network's design choices function as the SA metaheuristic's logic.

In the network design process, the network's fundamental structure must be determined. This decision primarily hinges on two approaches, considering whether the services should be centered around hubs. The first approach involves constructing a hub-centered network. In this thesis, hub-centered networks are defined as networks where all services include at least one hub, where a hub is a port that is part of 20 or more market orders. A market order is a port that serves as an origin- or destination port for more than 20 commodities. A visual representation of a hub-centered network can be seen in figure 3. The purpose is to center all services around at least one hub to minimize the transshipments needed to transport commodities.

The second alternative involves not designing the services around hubs. Instead, services are constructed in continuation of each other, forming a chain of services. This type of network is referred to as a conveyor belt network. Here, the services are not forced to include a hub in their itinerary and will, therefore, result in shorter services. On the other hand, this necessitates an increase in transshipments required to deliver cargo. A visual depiction of a conveyor belt network structure is illustrated in figure 4.

The method that will be considered in this thesis is the hub-centered network approach. This choice stems from prioritizing the reduction of transshipments over shorter services. The services within the hub-centered network typically fall into three distinct classes: simple service, butterfly service, and complex service (see figure 2). A simple service is a service that visits each port once, causing a circular itinerary. The butterfly services involve butterfly ports, which are allowed to be visited several times, thus connecting two or more "butterfly wings" consisting of ports. Lastly, complex services are services in which ports can be visited multiple times. The services addressed in this thesis are modeled as simple services where vessels are restricted to visiting ports only once. The decision to model the services as simple services is motivated by the aim to reduce the complexity associated with their creation.



(a) Simple service (b) Butterfly service (c) Complex service

Figure 2: Types of services: a) Simple service that visits every port exactly once. b) Butterfly service, where the center port C is visited multiple times. c) Complex service, where all ports can be visited multiple times (Karsten et al. 2017).

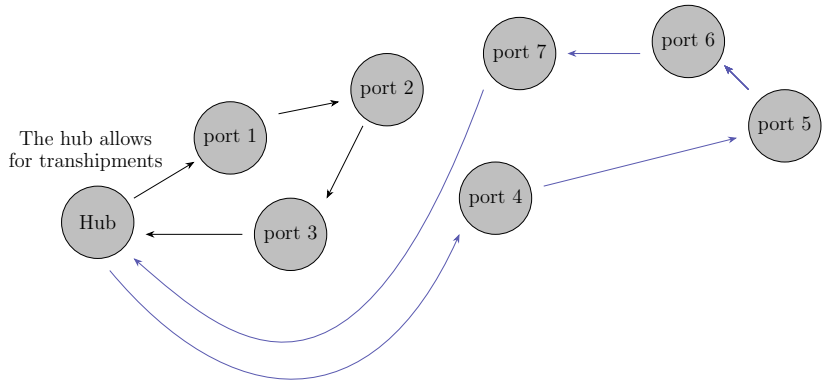


Figure 3: Hub-centered network: Each service is represented by a distinct color and is connected to a hub. The hub serves as the network's center, allowing convenient distribution of cargo. Note that the blue service is long but does not need transshipment for demands from and supplies to the hub.

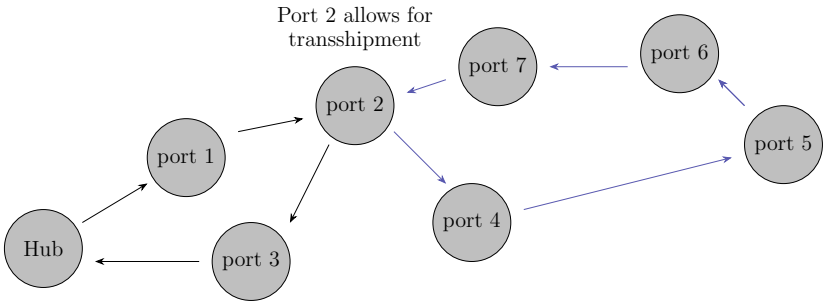


Figure 4: Conveyor network: Each service is represented by a distinct color. The services are linked, forming a conveyor belt network. As one service ends, the next one begins. Note that the blue service enables transshipments through port 2 for all commodities flowing to or from the hub.

4.1 Method for constructing services

Implementing a service network is based on designing a set of services and then evaluating and optimizing the services with a metaheuristic. The optimization of the services is based on an objective function that minimizes the total fixed costs of the services. Once the services are created, they will serve as the foundational network for solving the final MIP in section 6. The metaheuristic imposes a level of uncertainty when determining the effectiveness of the services created. This is because the metaheuristic only concerns the fixed costs of the services, while in practice, both the fixed- and variable transportation costs determine the effectiveness of the services. It is, therefore, uncertain whether a generated cost-effective service from the metaheuristic is effective in the final network after solving the MIP, as the variable costs are only implemented in the MIP.

Designing the service network is based on a list called *Ports*. *Ports* is an instance-based list comprising unique ports. The list *Ports* can be found in table 6.

Portindex	Port
1	port 1
2	port 2
...	...
n	port n

Table 6: A 2D-*Ports* array, with the unique ports of an instance, is created for each instance in the LINER-LIB dataset.

The vessels available for each instance are found in a list called *vessels*, depicted in table 7.

Vessel class	**Capacity FFE**	**TC rate daily (fixed Cost)**	**designSpeed**	**Bunker ton per day at designSpeed**	**Idle Consumption ton/day**
Feeder_450	450	5000	12	18.8	2.4
Feeder_800	800	8000	14	23.7	2.5
Feeder_450	450	5000	12	18.8	2.4
Feeder_800	800	8000	14	23.7	2.5
Feeder_450	450	5000	12	18.8	2.4
Feeder_450	450	5000	12	18.8	2.4

Table 7: The vessels array for the Baltic instance. It alternates between the v^F as long as the amount of a distinct v^F is not violated.

The construction of services is executed through an iterative process that involves

looping through the lists *Ports* and *Vessels*. Vessels are deployed iteratively to visit the ports in *Ports* sequentially and load cargo. An initial service is established when a vessel reaches its cargo capacity or predefined travel distance limit. This iterative process continues until either all vessels have been allocated to a service or until all existing ports within the instance are included in a service. Note that vessels never unload cargo during the creation of services; this is expanded upon in section 4.2.

The initial services are contained in a nested list called τ. The services $\in \tau$ are then further processed, ensuring that every port is visited by at least one service and all commodities have a feasible path from their origin port to their destination port and contain at least one hub. In addition, the services must be cyclic around the hub, meaning that the services must start and end in a hub. The tables 8 and 9 show the structure of the services and how they are contained in the nested list τ. The list *service* shows the structure of a service and will be modified during the service design to store the fixed cost of the service, its vessel class v^F, and the number of vessels the service requires such that all ports are visited once a week. These modifications are highlighted when made and are necessary as the services must carry this information for the second stage of the two-stage algorithm.

$service =$	$port_1$	$port_2$	...	$port_n$

Table 8: Structure of services in the τ.

$\tau =$	$service_1$	$service_2$	...	$service_i$

Table 9: Structure of τ.

A graphical example of how the algorithm for constructing the services is shown in figure 5 through 9. The example starts with the ports array seen in table 10 containing six ports, two of which are hubs.

Portindex	Port
1	Hub 1
2	Port 1
3	Port 2
4	Hub 2
5	Port 3
6	Port 4

Table 10: 2D-*Ports* array with ports present in the example

The example also includes three vessels that are used to construct services. These are referred to as *vessel 1, vessel 2, and vessel 3* for simplicity. It is important to note that only when a port has gotten all its supply picked up by one or more vessels the next port in the *ports* array is visited. In the example, each service is denoted by a distinct color.

- Figure 5 shows all unvisited ports before the construction of services is begun.
- In figure 6, the initial steps are commenced where *vessel 1* starts its service in *Hub 1* and picks up the supply at *Hub 1*. Then it sails to *Port 1* and picks up all corresponding supply. Due to the excessive distance to the subsequent port in table 10, *vessel 1* cannot travel to that port. Therefore, *vessel 2* is deployed.

  In step 2, *vessel 2* starts its service in *Port 2*, picks up all of the supply, and sails to *Hub 2*. *vessel 2* then picks up all supply at *Hub 2*, and does not have enough distance left to sail to the subsequent port, *Port 3*.

  In step 3, *vessel 3* starts its service in *Port 3*, picks up all of the supply, and cannot travel to *Port 4* due to the distance being too far. *Port 4* is therefore not added to a service via the vessels, and the service initiated by *vessel 3* consists only of *Port 3*.
- Figure 7 illustrates how ports, initially not assigned to any service, are included in the service of the closest port. Step 4 shows that the port closest to *Port 4* is *Port 3*, and therefore *Port 4* is added to the service of *Port 3*. Now, the current distance of the service exceeds the weekly travel limit of *vessel 3*. To meet the requirement of visiting ports every week, additional vessels of the same v^F will be deployed on the service at a later stage to ensure that this constraint is adhered to.
- Figure 8 shows how all services are forced to include a minimum of one hub and how they are made cyclic around that hub. The black and blue services already contain a hub. Thus, they are made cyclic by adding an arc between the last port, the corresponding vessel visits, and the hub of the service. The green service does not contain a hub and needs one inserted. This process involves identifying the hub nearest to one of the ports in the service. In this case, *Hub 2* is added to the green service due to its short distance to *Port 4* (see step 7). Step 8 then shows how the green service is made cyclic around *Hub 2*.
- Lastly, figure 9 shows how services are modified to allow for transshipment. The basic idea is to create a minimum spanning tree (MST) between the services and connect the services based on the MST. The blue and green services are already

connected at *Hub 2* and do not need modifications. The black service is not connected to the blue or green services and, therefore, needs to visit a port from one of them to allow for transshipments. The service that the black service should be connected to is determined by the MST - in this case, *Port 2* from the blue service is inserted in the black service. *Port 2* therefore facilitates transshipment between the black and blue services. The services are finished, and the example is done.

After completing the services in τ, the cumulative fixed cost of all these services will be computed, as detailed further in Section 4.3. To construct services with the lowest conceivable fixed costs, an optimization strategy is employed that specifically targets the fixed costs. This optimization is based on the SA metaheuristic, which aims to generate the services with the lowest fixed costs. Subsequently, this total fixed cost of τ serves as the objective value in the SA metaheuristic. For the SA metaheuristic to search for new improving services, two ports in the *Ports*-array swap indices in the array. This results in the next iteration of the SA metaheuristic, generating a new set of services as the vessels visit the ports in a new sequence. As previously mentioned, the SA metaheuristic produces a set of services for each instance. This set surpasses the number of services generated from a single run of the SA metaheuristic. The purpose behind constructing a larger set of services is to facilitate the optimization of the MIP. To create more unique services than a single run of the SA metaheuristic can construct, vessels' attributes capacity and travel distance range will be artificially modified. This is expanded upon in section 7.2. When solving the MIP, the objective is to reduce the costs of operating and flowing cargo through the network consisting of the set of services generated using the SA metaheuristic.

The SA metaheuristic has two hyperparameters that must be tuned. This tuning is performed by grid search, which will be described in section 7.1. With the tuned hyperparameters, the set of services is created for the second stage of the two-stage stage algorithm.

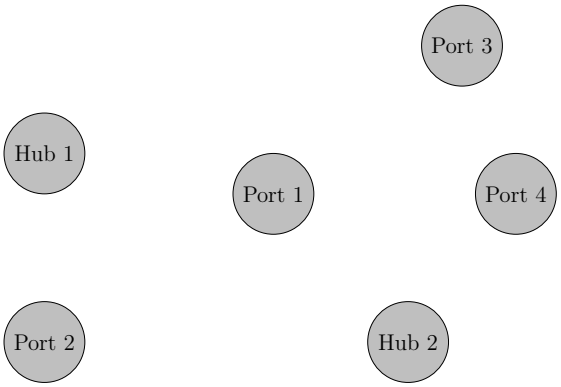


Figure 5: Visualization of ports present in the example.

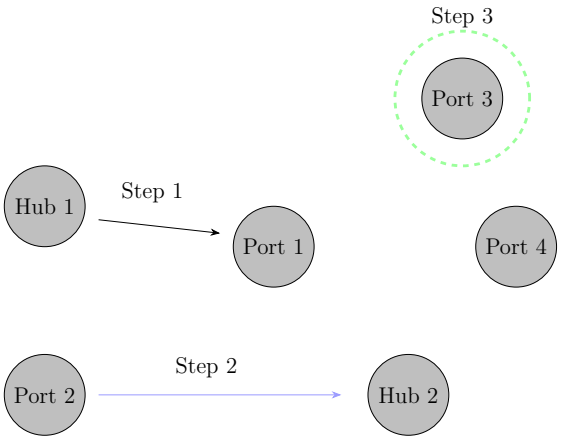


Figure 6: Initial services before making sure that all ports are present in at least one service

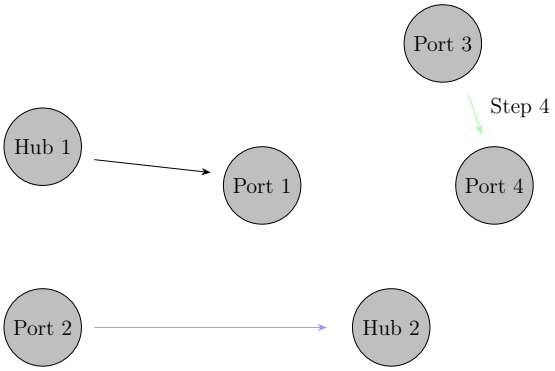


Figure 7: After making sure that all ports are present in a service

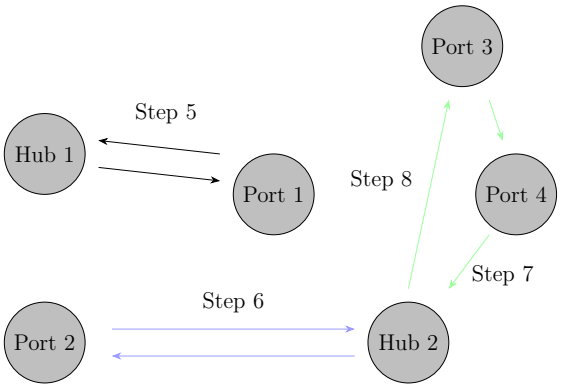


Figure 8: After making sure that all services contain at least one hub and making all services cyclic around a hub

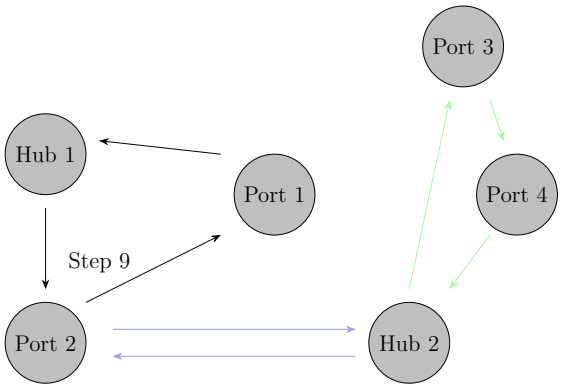


Figure 9: The finished services after making sure that all services are connected through transshipment nodes

4.2 Implementation of service design

The primary algorithm responsible for service creation is algorithm 1, referred to as *serviceslice*(*ports*, *vessels*). This function is designed to take two inputs: the first being the *ports*-array (as depicted in table 6), and the second input being the *vessels*-array (seen in table 7). The logic behind *serviceslice(ports, vessels)* depends on the capacities of vessels; the distance vessels can travel in a week, and the weekly supply of FFE at the ports. In algorithm 1, vessels never unload the FFE they pick up; the supply is therefore calculated by taking the average market order size of that particular port. The average market order size is an estimation of the supply of each port, which is based on the port's supply and demand. If one only considered the supply of a particular port, demand would be disregarded completely in the SND. This approach provides a reasonable estimation of how the services would function, considering the inherent constraints and complexities of the SND.

Figure 10 shows how the supply of two arbitrary ports (*Port 1 and Port 2*) are estimated. Here, *Port 1* needs to supply *Port 2* with 400 FFE and demands 300 FFE from *Port 2*. Thus, the average supply for *Port 1* is 350 FFE and thereby underestimated by 50 FEE. The opposite applies for *Port 2*. Here, the supply is overestimated by 50 FFE. Further, it might be the case that when solving the MIP, more commodities, through transshipments, may use the edges between *Port 1* and *Port 2*, meaning that the estimation of the needed capacity on the edges is off as well.

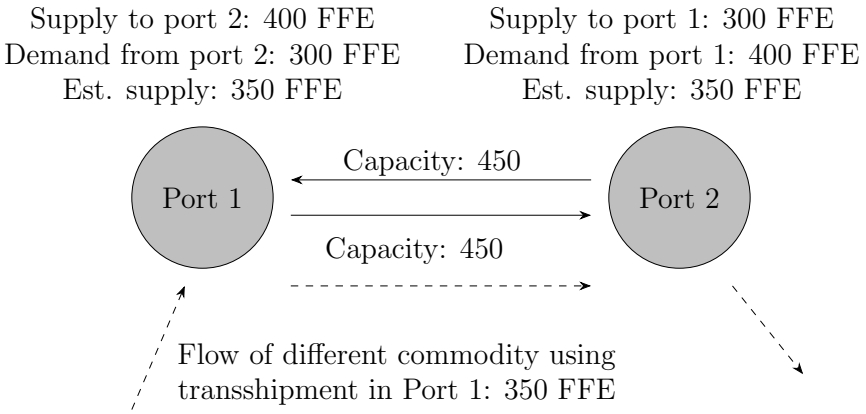


Figure 10: Visualization of how the supply of the ports can be over- and underestimated by taking the average of all market orders that a port is part of, as well as the estimation of capacity needed on edges between ports being off due to transshipment. The dashed arrow indicates an arbitrary transshipment.

In *serviceslice(ports, vessels)*, each available vessel, denoted as v_i, is associated with two constants utilized in the creation of services (*services* in algorithm 1): namely, v_i's capacity and the distance it can sail in a week. The *Ports*-array keeps track of the

subsequent ports to be visited. Let p_k denote the current port. If the supply of p_k has been covered by one or more vessels, port p_{k+1} is visited. *Serviceslice(ports, vessels)* essentially consist of six distinct cases, outlined below, each resulting in different outcomes for v_i and p_k. These cases are evaluated sequentially; if one case is irrelevant, the subsequent case is considered. The for-loop within *serviceslice(ports, vessels)* assesses these cases until either all ports have been visited or until all available vessels have been allocated to a service. The initial v_i commences its service at p_0 where p_0 is the first port in *Ports*. The cases below explain the different scenarios occurring within *serviceslice(ports, vessels)* during service construction (as annotated within algorithm 1).

- Case 1: If all ports have been visited, the for loop breaks. If the current service of v_i contains ports, append the service of v_i to τ.
- Case 2: Else if p_k is the first port that v_i visits and v_i possesses unused capacity greater than the supply available at pk, then v_i picks up all of the supply at p_k and visits p_{k+1}. Add p_k to v_i's service.
- Case 3: Else if p_k is not the first port visited by v_i and v_i possesses unused capacity greater than the supply available at pk and v_i can travel the distance to p_k. Then v_i picks up all of the supply at p_k and visits p_{k+1}. Add p_k to v_i's service.
- Case 4: Else if p_k is the first port that v_i visits and v_i possesses unused capacity less than the available supply at pk, vi loads cargo equivalent to its remaining capacity. v_{i+1} then starts its service in p_k. Add p_k to v_i's service, and add v_i's service to τ.
- Case 5: Else if p_k is not the first port that v_i visits and v_i possesses unused capacity less than the supply available at pk and v_i can travel the distance to p_k. Then v_i loads cargo equivalent to its remaining capacity and v_{i+1} starts its service in p_k. Add p_k to v_i's service, and add v_i's service to τ.
- Case 6: Else if v_i is not capable of reaching p_k, v_i appends its service to τ and v_{i+1} starts its service in p_k. Add v_i's service to τ.

As mentioned in section 4.1, all services created in the cases are added to the nested list τ, which is the primary data structure for storing the constructed services denoted as a set S. Then *serviceslice(ports, vessels)* saves the index of the last port visited and calls the functions *addport(τ,index,tour)*, *addhub(τ)* and *transship(τ)*. The index of the last port visited is used to add ports that have not yet been assigned to a service

in the function *addport(τ,index,tour)*. The finished services are returned when these function calls have been made in *serviceslice(ports, vessels)*.

Algorithm 1 Construct services

```
1:  procedure serviceslice(ports, vessels)
2:      τ = empty list                                   ▷ List of services
3:      k = 0                                  ▷ Tracks port to be visited next
4:      for each vessel in the vessels-array do
5:          service = empty list                          ▷ v_i's service

6:          while k ≤ length(ports)−1 do
7:              maxdist = distance v_i can travel

8:              if k== length(ports) then                          ▷ Case 1
9:                  if length(service) ≠ 0 then
10:                     append service to τ
11:                 end if
12:                 break

13:             else if length(service) < 1 and                    ▷ Case 2
                    supply of p_k ≤ v_i's unused capacity then
14:                 update capacity used for v_i
15:                 supply for p_k = 0
16:                 append p_k to service
17:                 k+=1
18:                 continue                              ▷ go to next port

19:             else if 1 ≤ length(service) and                    ▷ Case 3
                    supply of p_k ≤ v_i's unused capacity and
                    distance of service + distance to p_k ≤ maxdist then
20:                 update capacity used and distance of service for v_i
21:                 supply for p_k = 0
22:                 append p_k to service
23:                 k+=1
24:                 continue                              ▷ go to next port

25:             else if length(service) < 1 and                    ▷ Case 4
                    v_i's unused capacity < supply of p_k then
26:                 supply at p_k = supply at p_k − v_i's unused capacity
```

```
27:             v_i's unused capacity = 0
28:             append p_k to service
29:             append service to τ
30:             break                                          ▷ Go to next vessel

31:         else if 1 ≤ length(service) and                    ▷ Case 5
                v_i's unused capacity < supply of p_k and
                distance of service + distance to p_k ≤ maxdist then
32:             supply at p_k = supply at p_k − v_i's unused capacity
33:             v_i's unused capacity = 0
34:             update distance of service for v_i
35:             append p_k to service
36:             append service to τ
37:             break                                          ▷ Go to next vessel

38:         else if maxdist ≤ distance of service + distance to p_k then   ▷ Case 6
39:             append service to τ
40:             break                                          ▷ Go to next vessel

41:         end if
42:     end while
43: end for
44: index = index of last added port in ports-array
45: call function addport(τ, index, tour)
46: call function addhub(τ)
47: call function transship(τ)
48: return τ
49: end procedure
```

The function addport($\tau, index, tour$) seen in algorithm 2 ensures that all ports are present in at least one service. Ports might not be initially added to a service if the available vessels lack the capacity or cannot cover the required distance to visit all the ports in the given instance. This is achieved by creating two lists: the first, named *portsadded*, contains all ports that have been included in a service, while the second, named *portmiss*, encompasses all ports that are yet to be added to a service. The ports contained in *portsadded* are denoted as pa_g, and the ports contained in *portmiss* are denoted as pm_i. Then for each $\text{pm}_i \in$ *portmiss*, identify the closest $\text{pa}_g \in$ *portsadded* by using algorithm 3 (*distance(origin, destination)*), which calculates the distance between

two ports. Then, identify the first service containing pa_g and insert pm_i either before or after pa_g within that service such that the added distance of the insertion is minimized. This process continues until all ports in *portmiss* are included within a service, resulting in an updated τ list, which is then returned.

Algorithm 2 Add ports to services, if they were not previously added

```
1: procedure addport(τ, index, tour)
2:     portmiss = list of ports not added to a service

3:     for i in portmiss do
4:         portsadded = list of ports added to a service
5:         dist=inf
6:         PortAdd = empty string        ▷ Port that is closest to port that has not yet
           been inserted in a service

7:         for g in AddedPorts do
8:             if distance(pm_i,pa_g) < dist then
9:                 dist=distance(pm_i,pa_g)
10:                Portadd = pa_g
11:            end if

12:            for l in range(length(τ)) do
13:                find the first service in τ that contains the port PortAdd.
                   insert pm_i before or after PortAdd in τ_l
                   such that the added distance of insertion is minimized.
                   If the service τ_l consists of a single port,
                   insert pm_i before PortAdd.
14:            end for
15:        end for
16:    end for
17:    return τ
18: end procedure
```

Algorithm 3 Distance between two ports

```
1: procedure distance (origin, destination)
2:     return distance between origin and destination
3: end procedure
```

Hubs are implemented when all ports are present in at least one service. Hubs are crucial in ensuring the efficient flow of cargo and linking economies worldwide. These hubs serve as central points for shipping, tracking, and distributing cargo, so it moves smoothly between various regions (ShipsGo n.d.). All services in τ are forced to include at least one hub in the function *addhub(τ)* - see algorithm 4. The insertion of the hub happens such that the added distance of the insertion is minimized. The hub that is inserted is the hub that lies the closest to one of the ports in the service. Further, when a hub is inserted into the service, the service is made cyclic around that hub. Lastly, the v^F is inserted as the first element of the service - see new structure in table 11. Then, τ containing the updated *services* is returned.

$service =$	v^F	$port_1$	$port_2$	...	$port_n$

Table 11: Updated Structure of services in the τ.

Algorithm 4 Make sure that all services contain a hub and are cyclic

```
1:  procedure addhub(τ)
2:      for i in range length(τ) do
3:          if there is a hub in τ_i and length τ_i == 1 then
4:              append random port ≠ hub in τ_i to τ_i

5:          else if there is no hub in τ_i then
6:              find the hub closest to one of the ports in τ_i.
7:              insert the hub before or after the port closest by in τ_i such
                that the added distance of insertion is minimized
8:          end if
9:          make τ_i cyclic around the first hub in τ_i
            insert v^F as first element in τ_i
10:     end for
11:     return τ
12: end procedure
```

The final modification applied to the services within τ is to facilitate transshipments. This modification ensures the ability to move cargo from one service to another via an intermediary port, creating connectivity between services. The function *transship(τ)* ensures at least one transshipment is possible in each service. This is important as some commodities might not have a feasible shipping path from the origin to the destination port without transshipments. Transshipment facilitation is implemented by creating a MST between the services contained in τ. In addition, services will be represented by one node when creating the MST illustrated in figure 11.

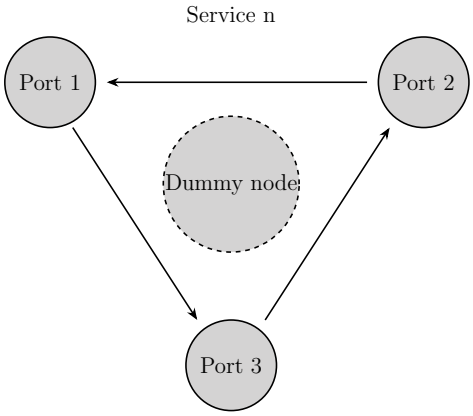


Figure 11: Dummy-variable for service coordinates.

To convert a service to a dummy node, an average of the longitude and latitude of the service's ports is calculated as seen below:

$$\text{Dummy Average Coordinates} = \left(\frac{1}{n} \sum_{i=1}^{n} \text{Longitude}_i, \frac{1}{n} \sum_{i=1}^{n} \text{Latitude}_i \right) \tag{1}$$

Then Kruskal's algorithm, found in algorithm 6, is applied to create the MST between the dummy nodes. The weight of the edges used in Kruskal's algorithm is the Euclidean distance between the dummy nodes calculated as:

$$\text{Euclidean Distance} = \sqrt{(\text{longitude}_2 - \text{longitude}_1)^2 + (\text{latitude}_2 - \text{latitude}_1)^2} \tag{2}$$

For each edge(u,v) between vertex u and v in the MST, a validation check is performed to determine if service$_u$ and service$_v$ already share a common port. If they do, no modifications are made to service$_u$ and service$_v$. However, if service$_u$ and service$_v$ lack a shared port, two ports are selected, one from service$_u$ and the other from service$_v$, such that these two ports are the ones closest to each other. These ports are denoted as $portU$ and $portV$, respectively. Then, a placeholder for each service is created. $PortU$ is inserted into the placeholder of service$_v$, and vice versa, ensuring the service remains cyclic around the hub while minimizing the added distance of the service. The placeholder service that gains the least added distance is utilized to update the original service in τ. Once the services facilitate transshipment, τ is returned with the design of the finished services.

Algorithm 5 Create transhipment-nodes in each service

1: **procedure** transship(τ)
2: calculate average XY-coordinates of each τ_i in τ
3: create MST using Kruskal's algorithm with the weight of edges being distance between average XY-coordinates of each τ_i in τ

4: **for** each edge(u,v) in mst **do**
5: **if** τ_u shares port with τ_v **then**
6: **continue**

7: **else**
find a port in τ_u and in τ_v such that these ports
are the ones closest to each other, and define them as:
8: portU = port from τ_u
9: portV = port from τ_v

10: **for** portU and portV **do**
11: placeholderU=τ_u
12: placeholderV=τ_v
13: **insert** portU before and after portV in placeholderV and vice versa,
and calculate the added distance of the insertion

14: **if** insertion happens at a hub **then**
15: then **insert** after hub at index 2 or before hub at index -2
▷ to keep the service cyclic around the hub
16: **end if**
17: **end for**
18: **end if**
19: make the insertion that added the least distance to the
placeholder on the original service in τ.
20: **end for**
21: **return** τ
22: **end procedure**

4.3 Fixed costs of services

As mentioned earlier, evaluating the set of services S constructed in the function *serviceslice(ports, ships)* is based on the total fixed cost, where minimization of the total fixed cost is favorable. The function *servicecost(τ)* seen in algorithm 7 calculates the total weekly cost of services in τ. The fixed cost of a single service consists of fixed port call costs, costs of having n vessels of v^F deployed on the service, the cost of bunker fuel consumed during sailing, and the cost of idling at ports. The formula for total fixed cost is found in equation 3.

Algorithm 6 Kruskal's algorithm (programiz n.d.)

```
1: procedure kruskal_algo(G)                    ▷ Where G is a directed graph
2:     A = ∅
3:     for each vertex v ∈ G.V do
4:         Make − Set(v)
5:     end for
6:     for each edge (u, v) ∈ G.E ordered by increasing order by weight (u, v) do
7:         if Find-Set (U) ≠ Find-Set (v) then
8:             A = A ∪ {(u, v)}
9:             UNION(u, v)
10:        end if
11:    end for
12:    return A
13: end procedure
```

$$\text{Total Fixed Cost} = \sum_{s \in S} \big(\text{Portcall Cost}_s + \text{Fixed Cost for v}^F{}_s \cdot n + \text{Fuel Cost}_s + \text{Idle Cost}_s\big) \tag{3}$$

Eq. 3 calculates the total fixed cost of the services $S \in \tau$. $\sum_s$ represents the sum over all services. The costs associated with the s-th service are Portcall Cost$_s$, Fixed Cost for Vessel$_s \cdot n$, where n is the number of vessels of class v^F, Fuel Cost$_s$, and Idle Cost$_s$.

Some assumptions need to be made to calculate the fixed cost of a service. The first assumption is a flat bunker price of 600 USD per ton. The second assumption is that a vessel spends, on average, 24 hours when calling a port. This is based on historical data regardless of vessel size (Brouer et al. 2014). The third assumption is that v^F sails at s^F_* irrespective of the possibility of faster or slower speeds. This might result in overestimating the number of vessels of class v^F deployed in services. The issue arises when an extra vessel is deployed in the service when the distance surpasses the distance a vessel can sail. Consequently, this means that if the distance of a service is only a bit longer than the distance a vessel of v^F can travel, an additional vessel is added.

Algorithm 7 Calculate total cost of τ

Fixed cost is calculated on a weekly basis
procedure servicecost(τ)
    cost = 0 ▷ initialize cost
    for i in range length(τ) **do**
        fixed_cost_vessel = $T^f \cdot 7$ ▷ for v^F on τ_i
        distance_service = 0
        portcost_service = 0

        for for k in range(1,length(τ_i)-1 **do**
            distance_service + = distance($\tau_{i,k}$, $\tau_{i,k+1}$)
            portcost_service + = f^P of $\tau_{i,k+1}$
        end for

        fuel_service = ton_fuel_per_knot · distance_service · 600 ▷ for v^F on τ_i at s^F_*
        distance_day = $24 \cdot s^F_*$ for v^F on τ_i
        vessel_dist_place = distance_service + (no. ports in τ_i-1)*distance_day
        no_vessels = $\lceil$ vessel_dist_place/(distance_day*7) $\rceil$
        idle_cost = f^F_0 · (no. ports in τ_i-1) · 600
        cost+ = portcost_service+(fixed_cost_vessel·no_vessels)+fuel_service+idle_cost
    end for
    return cost
end procedure

A similar function to algorithm 7 is implemented in algorithm 8 called *FixedCostServices(τ)*. The fixed cost calculations for each service in algorithm 8 mirror those outlined in algorithm 7. However, instead of taking the sum of the fixed cost for each service in τ, this function appends the individual fixed cost of each service and its corresponding number of vessels to the corresponding service in τ. The *FixedCostServices(τ)* function is called in the algorithm for the SA metaheuristic (see algorithm 9), and is called if the newly constructed set of services, S, is better than the current best set of services. This results in the final structure of each service seen in table 12.

$service =$	$No.\ of\ vessels\ of\ v^F$	$Fixed\ cost\ of\ service$	v^F	$port_1$	$port_2$	...	$port_n$

Table 12: Final structure of services for the MIP.

Algorithm 8 Calculate fixed cost for services

Fixed cost is calculated every week
procedure FixedCostServices(τ)
    for i in range length(τ) **do**
        cost = 0 ▷ initialize cost
        fixed_cost_vessel = $T^f \cdot 7$ ▷ for v^F on τ_i
        distance_service = 0
        portcost_service = 0

        for for k in range(1,length(τ_i)-1 **do**
            distance_service + = distance($\tau_{i,k}$, $\tau_{i,k+1}$)
            portcost_service + = f^P of $\tau_{i,k+1}$
        end for

        fuel_service = ton_fuel_per_knot $\cdot$ distance_service $\cdot$ 600 ▷ for v^F on τ_i at s_*^F
        distance_day = $24 \cdot s_*^F$ for v^F on τ_i
        vessel_dist_place = distance_service + (no. ports in τ_i-1)*distance_day
        no_vessels = $\lceil$ vessel_dist_place/(distance_day*7) $\rceil$
        idle_cost = $f_0^F \cdot$ (no. ports in τ_i-1) $\cdot$ 600
        cost= portcost_service+(fixed_cost_vessel $\cdot$ no_vessels)+fuel_service+idle_cost
        insert no_vessels as first element in τ_i
        insert cost as second element in τ_i
    end for
    return τ
end procedure

4.4 Simulated annealing

The SA metaheuristic is a simple metaheuristic that imitates basic principles from physics, where the analogy is that when the metal cools, the particles settle into fixed configurations. The SA metaheuristic can be interpreted as a controlled random walk in the search space, where new random solutions are generated in the neighborhood of the current solution. If the new solution is an improvement to the current solution, the new solution is accepted and becomes the current one. Suppose the new solution is worse or equal to the current solution. In that case, it is subjected to a probabilistic acceptance criterion called the metropolis function, which depends on the magnitude of the deterioration and a search control parameter called temperature (Delahaye, Chaimatanan,

and Mongeau 2018). The metropolis function can be seen in eq. 4.

$$P(\Delta E, T) = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{-\frac{\Delta E}{T}} & \text{if } \Delta E \geq 0 \end{cases} \tag{4}$$

where ΔE represents the absolute difference between the current and new solutions, and T represents the current temperature. If the new solution is better than the current solution, the probability of it becoming the current solution is 1. If the new solution is worse or equal to the current solution, then the new solution has a probability of $e^{-\frac{\Delta E}{T}}$ of becoming the current solution.

As time passes, the temperature slowly decreases, meaning that non-improving solutions have a lower and lower chance of being accepted as new current solutions. Consequently, less and less hill-climbing is allowed to escape local extrema and continue the search for a global extrema. In terms of the analogy to physics, the "particles" fixates. (Crainic and Gendreau 2021). The flowchart for the SA is found in figure 12.

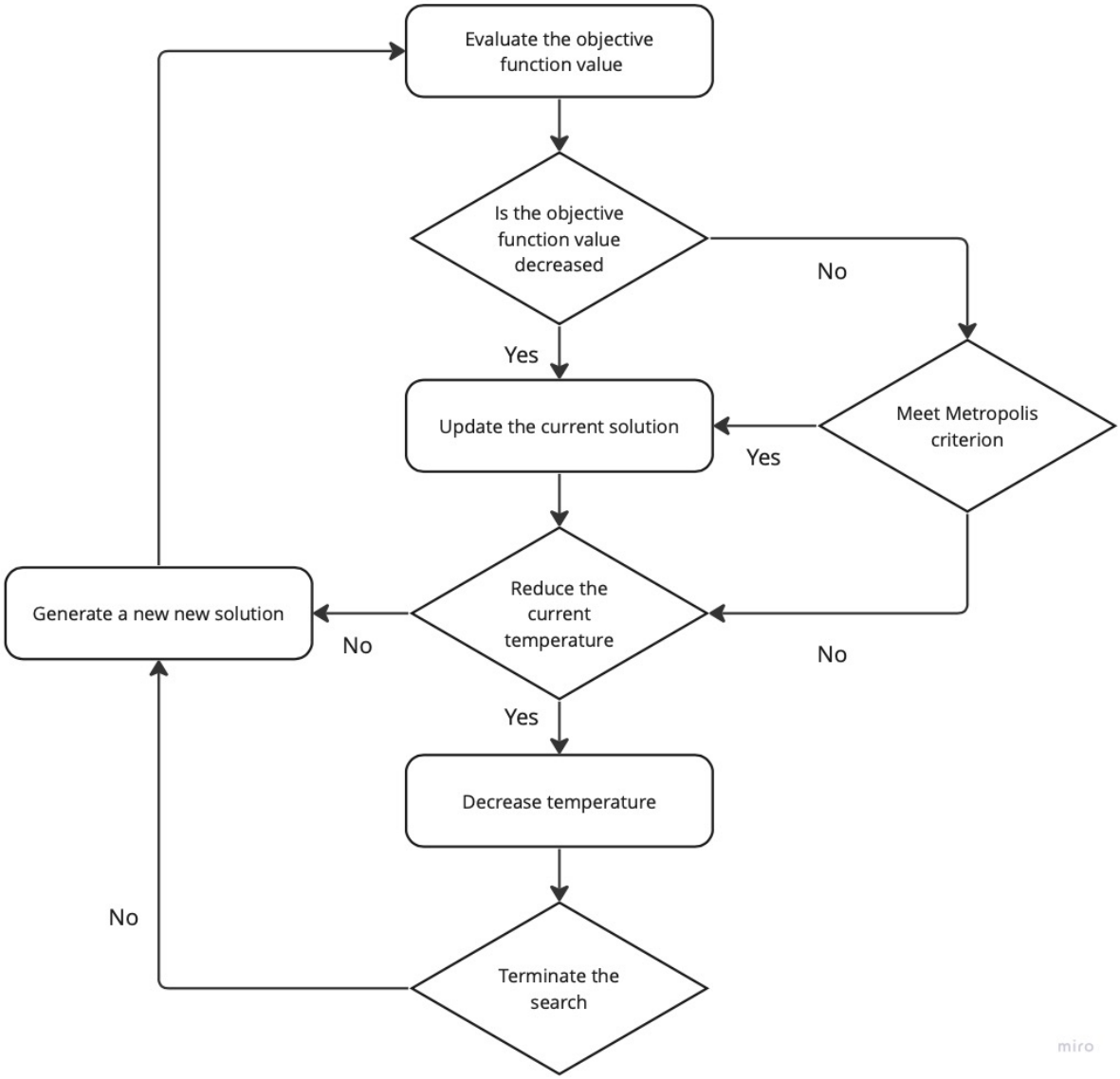


Figure 12: Flowchart of SA metaheuristic.

The implementation of the SA metaheuristic (algorithm 9) starts by initializing both a variable called bestsol, which contains the total fixed cost of the τ and a list containing the services with the lowest fixed costs. The temperature at time t is defined as a vector of n logarithmic spaced intervals $\in (tempend, temp0)$. An illustration of a logarithmic spaced is depicted in figure 13. The termination criterion for the SA implementation is met after completing 5000 iterations, and bestsol containing the cheapest τ is returned. The structure of each service in bestsol is shown in table 12, which will be used as input for the MIP. The convergence of a single run of the SA metaheuristic can be seen in figure 14. As the temperature decreases, the probability of accepting a worse or equal solution compared to the current solution also decreases. This is apparent by the downward slope depicted in the graph shown in figure 14.

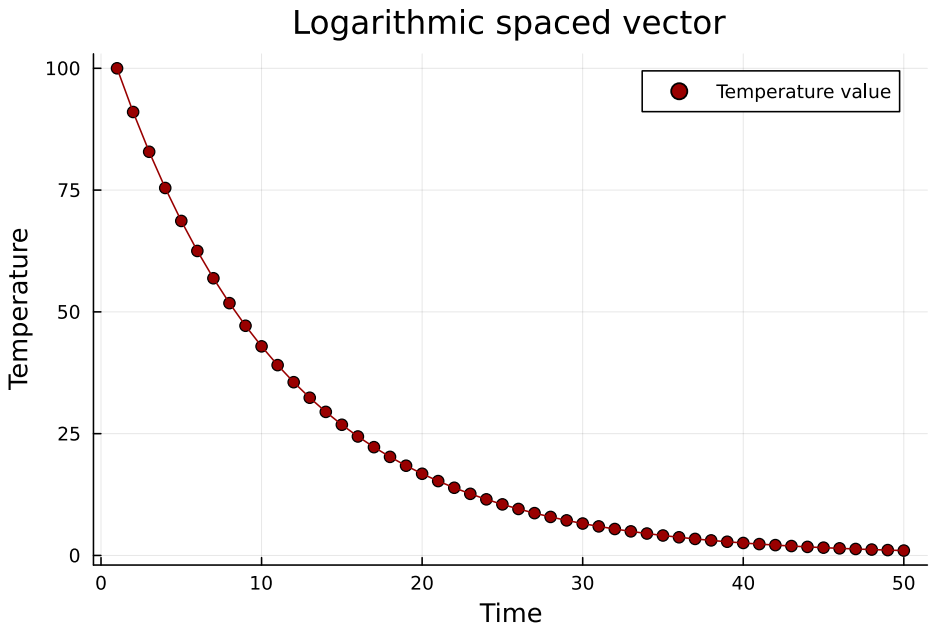


Figure 13: Logarithmic spaced vector with 50 intervals. The starting temperature is 100, and the end temperature is 0. The interval size between the points gets smaller and smaller as time passes.

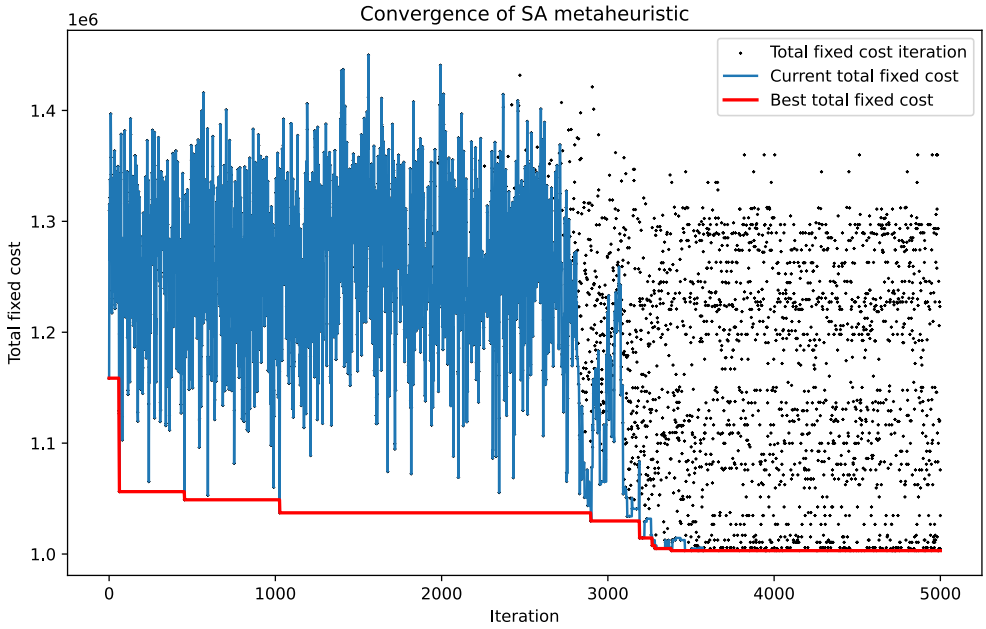


Figure 14: The convergence of an arbitrary run of the SA metaheuristic (5000 iterations). *Total fixed cost iteration* is the total fixed cost of the services found in the current iteration. *Current total fixed cost* is the solution accepted as the new current cost of the SA algorithm. *Best total fixed cost* is the set of services with the lowest total cost constructed.

Algorithm 9 Simulated annealing

currentserv=serviceslice(ports, vessels) ▷ Initial services created
bestsol = servicecost(currentserv) ▷ initialize variables
bestserv = FixedCostServices(currentserv)
currentsol=servicecost(currentserv)

for for temp in a logarithmic spaced vector $(temp0, tempend)$ **do**
    updateports = ports-array where two ports have switched position at random
    newservices = serviceslice(updateports, vessels)
    updatesol = servicecost(newservices)
    rand = random float $\in (0, 1)$

    if updatesol-currentsol< 0 **then**
        ports = updateports
        currentsol=updatesol

        if updatesol-bestsol < 0 **then**:
            bestsol = updatesol
            bestserv = FixedCostServices(newservices)
        end if

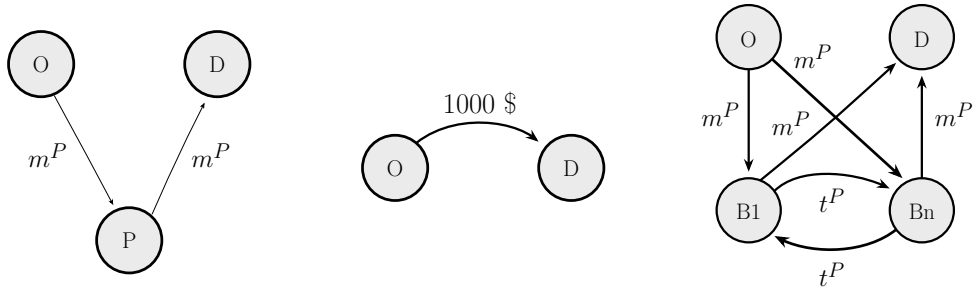
    else if $e^{-|(currentsol-updatesol)|/temp} \geq rand$ **then**
        tour = updateports
        currentsol=updatesol

    else if $e^{-|(currentsol-updatesol)|/temp} \leq rand$ **then**
        continue
    end if
end for
return bestserv

5 Network

In the domain of shipping networks, the physical property of shipping containers from A to B depends on several operational activities, such as (un)loading cargo and cargo transshipment. The optimization of these operational activities is paramount for achieving low costs and high delivery rates. Building upon the service design presented in

section 4, it becomes imperative to implement network modifications that consider the aforementioned operational activities. These modifications enable a more precise representation of container movements and their associated expenses. Throughout this section, the nodes in the services are referred to as ports, while the newly added modification nodes, to represent port activities, will be referred to simply as "nodes."



(a) Port configuration (b) Penalty arcs (c) Transshipment configuration

Figure 15: Illustration of network modification.

5.1 Representation of port activities

In constructing an accurate representation of shipping activities, the arrangement and orientation of the nodes and arcs, respectively, are crucial for ensuring a cohesive network that cannot be violated in terms of negative cycles regarding cost when flowing containers. The operations are categorized into three components: (un)loading containers, forfeiting containers, and transshipping containers. Each activity corresponds to a distinct setup of nodes and arcs within the network, as depicted in figure 15. These configurations are then implemented across designated ports within the services. Before implementing configurations to the ports, it is necessary to determine the transshipment ports, which are the ports visited by two or more services. Identifying these requires examining each service and evaluating whether two or more services intersect at the same ports to configure transshipment facilitation correctly.

The first configuration illustrated in subfigure (a) of figure 15 considers the moving costs, m_P, per FFE of (un)loading cargo at the ports. This configuration involves connecting two nodes to each port that does not offer transshipment. The first node is an origin node denoted as O with a directed arc pointing to the port P, and the other node is a destination node, D, connected with an arc pointing from P to D. The cost on each arc equals m_P, which is predetermined for each port $p \in P$ in LINER-LIB.

The second configuration comprises penalty arcs as illustrated in subfigure (b) of figure 15. These arcs impose a financial penalty of $1,000$ USD for failing to transport a

single FFE from its origin port to its destination port (Brouer et al. 2014). The configuration involves adding an arc from each origin port that points to the corresponding destination port for all commodities. This arc is not constrained by capacity limitations imposed by vessels. Instead, it solely serves as the physical property of being penalized for forfeiting containers.

The final port configuration that will be presented involves the modeling of transshipments. Transshipment refers to the process of transferring containers from one vessel to another. This transfer occurs at an intermediate port between two or more services where containers are unloaded from one vessel, stored in the terminal, and loaded onto another. To facilitate transshipment properly, a new node is introduced for each service that visits an intermediate port. This node symbolizes a dedicated berth exclusively allocated for that specific service. Thus, a transshipment port comprises berths designated for (un)loading containers for each service the port serves. As the transshipment ports are already determined, the transshipment representation can be constructed by introducing two bidirectional arcs, thus connecting all berths with all other berths. This causes a quadratic growth in the number of arcs as the number of berths increases and can be expressed as:

$$|T_p| = |B_p| \cdot (|B_p| - 1) \tag{5}$$

where $|T_p|$ is the number of transshipment arcs at port p and $|B_p|$ is the number of berths at port p. The cost of traversing these bidirectional arcs equals the transshipment cost of the corresponding intermediate ports, denoted as t^P. All berths are additionally connected with a source and a destination node, a connection that is equivalent to the one presented in subfigure (a) of figure 15, including the moving costs m^P.

The network configurations presented above will collectively constitute a graph $G(N, A)$ that is unique for each instance in LINER-LIB. Here, N denotes the set of nodes, including the ports from the services and the corresponding configuration nodes. A denotes the arcs, i.e., the possible sailings between the nodes N. The graphs serve as a comprehensive tool for modeling container flow dynamics, enabling an accurate representation of port operations. In addition, $G(N, A)$ will be used as the foundational network for flowing commodities from their origin port to their destination in the MIP presented in section 6.

An illustration is presented in figure 16 to visualize how the different port representations function in $G(N, A)$. This Illustration is based on the following services presented solely for demonstrative purposes:

Service 1: DEBRV $\rightarrow$ SEGOT $\rightarrow$ DKAAR $\rightarrow$ DEBRV
Service 2: DEBRV $\rightarrow$ DKAAR $\rightarrow$ RUKGD $\rightarrow$ DEBRV

According to the services, both DEBRV (Bremerhaven, Germany) and DKAAR (Aarhus, Denmark) must offer transshipment as they are intermediate ports between the two services.

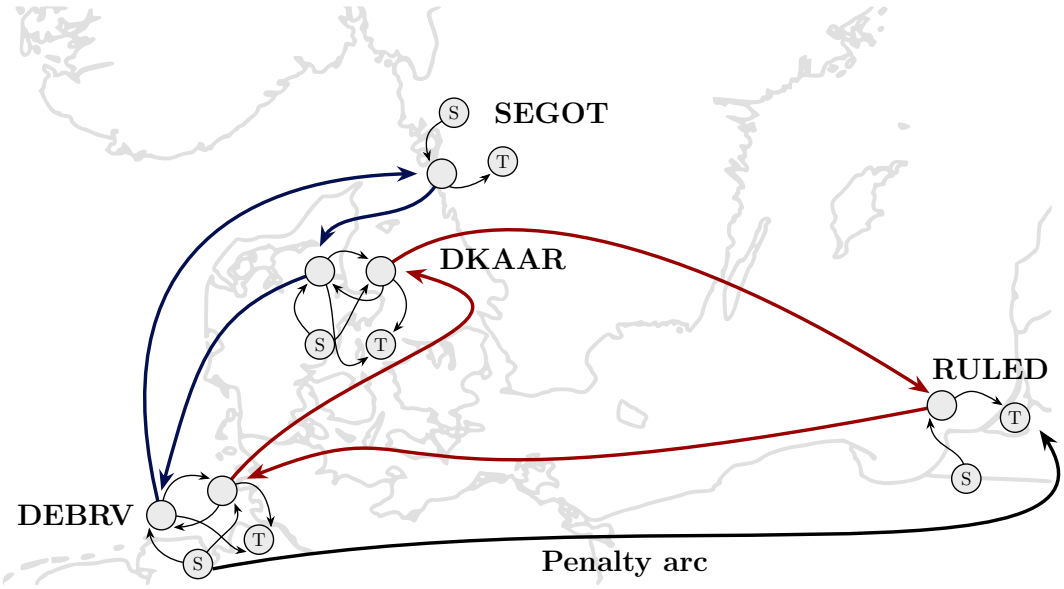


Figure 16: Illustration of $G(N, A)$ for Baltic. The blue nodes represent the rotation of service 1, while the red nodes constitute service 2. The thick black line is a penalty arc going from a source node to a destination node.

6 Mathematical model

This section introduces the MIP formulation for the second stage of the LSNDP, referred to as *Liner Shipping Multi-Commodity Flow problem* (LS-MCFP). The objective of the LS-MCFP is to determine the quantity of commodity $k \in K$ to be sent along a network from an origin port α_k^C to a destination port β_k^C while adhering to the vessel capacities of the services and minimizing overall transportation costs. Here, K is the set of all commodities of the cargo demand list C in section 3. The objective also encompasses identifying the services that optimize container flow. Consequently, the LS-MCFP will selectively choose a subset of the services generated in section 4 to be employed for flowing containers. The flow problem also considers a specified fleet denoted as the set V, comprising various vessel classes, v^F, with a corresponding capacity c^F. Furthermore, there is a limited availability of vessels within each vessel class, quantified as m_{v^F} for every $v^F \in V$.

In the context of the network, β_k^C demands a certain amount of commodity k, represented by q_k^C, measured in FFE. Conversely, α_k^C has a corresponding supply of

commodity k to be send to β_k^C. The intention is to send as much commodity from α_k^C to β_k^C as possible concerning vessel capacities. The network is represented by graph $G(N, A)$ introduced in section 5. Here, each arc $(i, j) \in A$ has an associated cost per FFE denoted as c_{ij} along with a capacity u_{ij}^s equal to the capacity c^F of v^F sailing in service s. In addition, all services have a corresponding fixed cost c_s, calculated in section 4.3.

6.1 Arc formulation

The properties of the LS-MCFP enable the formulation of the problem into an arc-based structure for which the commodity flow is determined on an arc basis. Let $G(N, A)$ be a directed graph formed by a set of services S along with the port modifications described in section 5. Moreover, a continuous variable x_{ij}^k is introduced, denoting the flow of commodity $k \in K$ transported by service $s \in S$ on arc $(i, j) \in A$. The aforementioned demand q_k^C for each β_k^C will be defined as:

$$\xi_i^k = \begin{cases} q_k^C & \text{if port}_i = \alpha_k^C \\ -q_k^C & \text{if port}_i = \beta_k^C \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

to support flow conservation in the formulation. The binary variable y_s denotes whether a service is included or excluded in the solution. The arc formulation of the LS-MCFP can be expressed as follows:

$$\min \quad \sum_{s \in S} c_s y_s + \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{7}$$

s.t.

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = \xi_i^k \qquad \forall i \in N, k \in K \tag{8}$$

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}^s y_s \qquad \forall s \in S, (i, j) \in A \tag{9}$$

$$\sum_{s \in S} m_{v^F}^s y_s \leq m_{v^F} \qquad \forall v^F \in V \tag{10}$$

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

$$y_s \in \{0, 1\} \qquad \forall s \in S \tag{12}$$

The objective function (eq. 7) minimizes the fixed- and variable transportation costs minus the total revenue for the transported commodities in FFE. The fixed costs encompass operational expenses of the services, such as TC rate, bunker consumption costs, and port call costs. The variable costs, measured on an FFE basis, comprise

transshipment costs within ports, the cost incurred when forfeiting containers, and moving costs in terms of (un)loading containers. The solution to the flow problem determines the optimal number of containers to send along each arc. These containers are transported utilizing the services associated with the arcs. The first constraint (eq. 8) ensures flow conservation, meaning that a port's total ingoing container flow equals the total outgoing container flow, thereby preventing containers from becoming trapped in certain ports without reaching their intended destination. Additionally, the constraint ensures that origin ports release the available supply while destination ports acquire the corresponding demand. This conservation principle ensures a balanced flow in the network. Constraint eq. 9 ensures that the total flow of containers along an arc does not exceed the capacity of the associated vessel class traversing that particular arc. The sum of vessels of class v^F employed in a service s is limited to m_{v^F} available vessels. This constraint is integrated in eq. 10.

The network offers a large number of arcs going from one port to another, serving as variables in the arc-formulated version of the LS-MCFP. The combinatorial complexity makes the model intractable to solve, especially in cases where many competing variables cannot satisfy the restrictions of the problem at once (Gilmore and Gomory 1961). In practice, only a subset of all the conceivable commodity flows is needed to achieve an optimal solution. Thus, only a limited number of variables contribute to solving the model to optimality. To overcome the computational demand of defining all conceivable commodity flows to solve the model, a common approach is applying Dantzig-Wolfe decomposition, which, by using column generation, solves a subproblem of the LS-MCFP (Ahuja, Magnanti, and Orlin 1993). Dantzig-Wolfe decomposition involves dividing the original problem into two components. The first component for solving the LS-MCFP is the master problem containing all possible variables in basis. Nested within the master problem lies the concept of the RMP. The RMP functions similarly to the master problem. However, the subset-oriented approach streamlines the computational process by utilizing a restricted set of variables yet aims to attain the same optimal solution as the master problem. The second component involves a subproblem incorporating dynamic column generation to enhance the RMP's objective function (Dai, Sun, and Wandelt 2016). Within this process, the dual problem of the RMP is solved to obtain dual variables, which are crucial for computing a reduced cost. This reduced cost determines the inclusion of specific variables into the model's basis, thereby optimizing and improving the overall objective function of the RMP.

The arc formulation can be reformulated using path flows with the Dantzig-Wolfe decomposition. Here, a path is represented as a sequential set of traversed arcs that brings a certain amount of commodity k from α_k^C to β_k^C. The cost of a particular path

is derived from the cumulative cost of the traversed arcs, resulting in a more refined representation of the variables that reduces the size of the master problem.

6.2 Path formulation

In the path formulation, a new continuous variable, x_p^k, is introduced, which represents a path $p^k \in \Omega^k$ flowing commodity k, where the set Ω^k for $k \in K$ contains all feasible paths for the corresponding commodity. This new variable denotes the flow of commodity k transported along a path in FFE. By addressing certain constraints and notations from the arc formulation, the LS-MCFP can readily be translated into the path formulation. This pertains to the flow conservation parameter ξ_i^k, defined in eq. 6, which sign will no longer depend on the i'th port in the flow conservation. Hence, it will be denoted as ξ^k and represents the demand in FFE of commodity k. Moreover, a new binary coefficient $a_{p,ij}^k$ is introduced. This coefficient equals 1 if path x_p^k uses arc (i, j), otherwise 0. The path formulation is as follows:

$$\min \quad \sum_{s \in S} c_s y_s + \sum_{k \in K} \sum_{p^k \in \Omega^k} c_p^k x_p^k \tag{13}$$

s.t.

$$\sum_{p^k \in \Omega^k} x_p^k = \xi^k \qquad \forall k \in K \tag{14}$$

$$\sum_{k \in K} \sum_{p^k \in \Omega^k} a_{p,ij}^k x_p^k \leq \sum_{s \in S} u_{ij}^s y_s \qquad \forall (i,j) \in A \tag{15}$$

$$\sum_{s \in S} m_{v^F}^s y_s \leq m_{v^F} \qquad \forall v^F \in V \tag{16}$$

$$x_p^k \geq 0 \qquad \forall k \in K, p^k \in \Omega^k \tag{17}$$

$$y_s \in \{0, 1\} \qquad \forall s \in S \tag{18}$$

Here, the objective (eq. 13) remains the minimization of both fixed and variable transportation costs minus the total revenue for the delivered commodities. The variable transportation costs encompass the cumulative flow costs incurred by traversing the arcs described in section 5 within the paths. This includes the moving cost of (un)loading containers, the cost of transshipping containers, and containers flowing on penalty arcs. Constraint eq. 14 is the flow conservation constraint ensuring that all demands are accommodated for each commodity k. The flow capacity constraint, eq. 15, functions as an upper bound on the amount of commodity flowing through arc (i, j). The vessel capacities u_{ij}^s on the corresponding service s, navigating through that arc, remain unviolated. Constraint eq. 16 is equivalent to eq. 10 presented in the arc formulation.

6.3 Column generation

Column generation is a mathematical optimization technique used to solve large-scale combinatorial problems. The method is particularly effective when dealing with problems that involve a large number of variables, but only a small subset of these variables are relevant for solving the model to optimality. This subset will be denoted as $\bar{\Omega}^k \subseteq \Omega^k$, to which the columns, i.e., paths generated by column generation, are added. Column generation can be referred to as the subproblem of the master problem since it iteratively identifies new paths from origin port α_k^C to destination port β_k^C for each commodity k. To determine whether the new feasible paths enter basis or not, the dual problem of the RMP must be solved, whereby the current optimal objective value and the dual variables are derived. Note that the binary variables y_s are relaxed to linear inequalities to solve the RMP as a linear program. The dual variables represent the shadow prices associated with the flow conservation constraints (eq. 14) for each commodity, denoted as π_k, and the capacity constraint of the network (eq. 15) denoted as λ_{ij}. Specifically, suppose there is an increase in the capacity of an arc or a change in the flow of a commodity. In that case, the dual variables indicate the change in the objective function value per unit increase (Sakarovitch 1983).

By possessing the dual variables, the reduced costs of the new candidate paths are calculated as follows:

$$\bar{c}_p = \sum_{(ij) \in p^k} (c_{ij} - \lambda_{ij}) - \pi_k \tag{19}$$

The determination of the reduced cost involves employing Dijkstra's algorithm, which can be found in algorithm 10, to identify the shortest path from α_k^C to β_k^C for each commodity k. Instead of traditional distance-based weights between ports, this approach utilizes the arc costs $c_{ij} - \lambda_{ij}$ as the weights on the arcs. As a result, the shortest path represents the most cost-effective route from an origin port to a destination port for a single container.

Algorithm 10 Dijkstra's shortest path

Require: $G(N, A)$, source s, destination t;

$Q \leftarrow \emptyset$
$y_s \leftarrow 0; y_i \leftarrow \infty$ for all other i
while $Q \neq \emptyset$ **do**
    select an $i \in Q$ minimizing y_i
    remove i from Q
    for $(i, j) \in A : j \notin Q$ **do**
        if $y_i + w_{ij} < y_j$ **then**
            $y_j \leftarrow y_i + w_{ij}$
            $p_j \leftarrow i$
        end if
    end for
    $S \leftarrow \emptyset$ ▷ Set of traversed ports
    $i \leftarrow t$
    while $p_i \neq \varnothing$ **do**
        insert i at the beginning of S
        $i \leftarrow p_i$
    end while
end while
return S, y_t

Prior to running Dijkstra's algorithm, the cost of each arc $(i, j) \in A$ is adjusted by subtracting λ_{ij} from the previous iteration, refining the path's cost. Then the reduced cost is calculated by subtracting the corresponding dual variable π_k for commodity k from the shortest path $\sum(c_{ij} - \lambda_{ij})$. If paths with negative reduced costs are identified, they are added to the RMP. It is crucial to emphasize that the capacity limit of vessels affects only specific arcs due to unavoidable port modifications. Those arcs not reliant on vessel capacities, such as transshipment arcs and origin/destination arcs, will have their corresponding λ_{ij} adjusted to 0 for practicality when calculating the reduced costs. Meanwhile, the λ_{ij} associated with arcs utilized by the services will remain unaltered. λ_{ij} is therefore expressed as:

$$\lambda_{ij} = \begin{cases} \lambda_{ij} & \text{if arc } (i, j) \in A \cap (i, j) \in S \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

The entire procedure of iteratively finding new paths, calculating the reduced costs, and adding the paths with negative reduced costs to the basis continues until the RMP is solved to optimality. The optimality criterion for the RMP is when no new paths

with negative reduced costs are found when solving the subproblem. The algorithm for the column generation that solves the subproblem can be found below:

Algorithm 11 Column Generation

Require: $G(N, A)$, initial columns $\bar{\Omega}^k$;

repeat
  solve RMP with initial basis $\bar{\Omega}^k$
  update $G(N, A)$ w.r.t. π_k and λ_{ij}
  $candidatepaths \leftarrow \emptyset$
  for $k \in K$ **do**
    $p^k \leftarrow Dijkstra(G(N, A), \alpha_k^C, \beta_k^C)$ ▷ Shortest path for each k
    insert p^k into $candidatepaths$
  end for
  $\bar{c}_p = \sum_{(i,j)\in p^k}(c_{ij} - \lambda_{ij}) - \pi_k$
  for $p^k \in candidatepaths[-\bar{c}_p]$ **do** ▷ Candidate paths with reduced cost
    insert p^k into $\bar{\Omega}^k$
  end for
until no path has been added

An initial solution containing a set of feasible paths is required for the RMP to request new columns from the subproblem dynamically. The initial solution must be chosen with consideration, ensuring simultaneous adherence to both the flow conservation constraint and the capacity constraint of the path formulated LS-MCFP. The penalty arcs presented in section 5 will be considered to achieve this. This is because the penalty arcs do not interfere with vessel capacities while facilitating the transport of all containers to their designated destination port, making them a good fit for the model. Recall from section 5 that the containers are not physically transported to their final destination. Instead, this method represents forfeited containers that cannot be delivered.

When the optimality criterion of the RMP is met, the paths contained in basis of the RMP are used to solve the LS-MCFP. When solving the LS-MCFP, the variable y_s, which was previously relaxed to solve the dual problem, will be altered back to a binary variable. As a result, fractional services become prohibited; each service is either entirely included or excluded from the final solution. This change will notably affect the final solution since the upper limit on the container flow of the arcs will no longer be perfectly constrained.

7 Computational experiments

This section will present the results of the computational experiments performed on the Baltic, WAF, and Mediterranean instances from the LINER-LIB benchmark suite.

7.1 Grid search for hyperparameter tuning in SA

The performance of the SA metaheuristic depends on the probability of accepting a new solution as its current solution, which is dependent on the temperature value. The higher the temperature, the more likely the metaheuristic is to accept a worse or equal solution as the current solution and thus allow for hill-climbing to escape local minima. From eq. 4 in section 4, it is seen that the temperature essentially becomes a variable that determines the trade-off between exploration and exploitation. The higher the temperature, the more explorative the model is. As the temperature decreases, the probability of accepting worse or equal solutions as the current solution decreases concurrently.

Optimizing the initial and final temperature values (temperature schedule) is crucial due to the significant impact the temperature has on the behavior and performance of the SA metaheuristic. The optimization method chosen for optimizing the temperature encompasses a simple grid search. Grid search is a method that allows for exploring the temperature domain. As the name indicates, the grid search creates a grid of different combinations of temperature schedule values. The temperature schedule that perform the best are the temperatures that will be used in the SA metaheuristic to construct the services used in the LS-MCFP. The grid of temperatures is seen in table 13.

$[10^{10}+1, 0]$	$[10^{9}+1, 0]$	...	$[10^{4}+1, 0]$	$[10^{3}+1, 0]$	$[10^{2}+1, 0]$	$[10^{1}+1, 0]$
$[10^{10}+1, 1000]$	$[10^{9}+1, 1000]$	...	$[10^{4}+1, 1000]$	$[10^{3}+1, 1000]$	-	-
$[10^{10}+1, 10000]$	$[10^{9}+1, 10000]$	...	$[10^{4}+1, 10000]$	-	-	-
$[10^{10}+1, 20000]$	$[10^{9}+1, 20000]$	...	$[10^{4}+1, 20000]$	-	-	-
$[10^{10}+1, 50000]$	$[10^{9}+1, 50000]$	...	$[10^{4}+1, 50000]$	-	-	-

Table 13: Grid-search temperatures - $[temp0, tempend]$. The empty fields are empty because the start temperature is required to be higher than the end temperature. The "+1" is added because the implementation of the SA metaheuristic relies on creating 5000 intervals between the starting and ending temperatures of the algorithm. If these temperatures are the same, then the intervals cannot be formed.

The grid search is implemented by using each grid field five times as the input for the SA metaheuristic. Subsequently, the average of the five results for each grid field

is computed. The best set of temperature values is determined by the lowest average total fixed cost across five runs.

The SA metaheuristic runs 5000 iterations for each iteration of the grid search. The average total fixed cost of the constructed services in each iteration is calculated to accurately estimate the true performance of the SA metaheuristic given the grid fields. If more than one grid field is optimal, then the uppermost left field is chosen.

The results of the grid for the Baltic instance are shown in table 14. The best set of temperature values is achieved when the temperature schedule is $[10^9 + 1, 0]$, resulting in an average total fixed cost of $984,898.2$ USD (see table 14).

	$\mathbf{10^{10}+1}$	$\mathbf{10^{9}+1}$	$\mathbf{10^{8}+1}$	$\mathbf{10^{7}+1}$	$\mathbf{10^{6}+1}$	$\mathbf{10^{5}+1}$	$\mathbf{10^{4}+1}$	$\mathbf{10^{3}+1}$	$\mathbf{10^{2}+1}$	$\mathbf{10^{1}+1}$
0	101,21	100,00	102,27	101,20	100,49	101,60	101,22	100,66	103,42	103,42
1000	101,40	100,00	102,16	101,49	100,57	101,99	101,26	100,34	-	-
10000	105,13	104,89	104,13	104,33	104,23	103,86	104,57	-	-	-
20000	104,34	103,50	104,89	104,48	104,63	104,87	-	-	-	-
50000	105,18	103,01	103,28	103,83	104,83	103,27	-	-	-	-

Table 14: Results of grid search for Baltic. Each cell represents the percentage difference from the best total average fixed cost.

For the WAF instance, the grid search identifies the set of temperatures as $[10^4+1, 0]$ with an average fixed cost of $4,198,157$ USD.

	$\mathbf{10^{10}+1}$	$\mathbf{10^{9}+1}$	$\mathbf{10^{8}+1}$	$\mathbf{10^{7}+1}$	$\mathbf{10^{6}+1}$	$\mathbf{10^{5}+1}$	$\mathbf{10^{4}+1}$	$\mathbf{10^{3}+1}$	$\mathbf{10^{2}+1}$	$\mathbf{10^{1}+1}$
0	102,51	104,44	101,70	103,79	101,35	100,73	100,00	107,33	107,81	105,83
1000	103,27	104,50	101,70	103,79	101,35	100,73	100,00	108,18	-	-
10000	115,95	115,11	115,32	118,66	116,76	117,26	116,76	-	-	-
20000	116,09	115,70	115,40	114,54	116,27	116,88	-	-	-	-
50000	119,11	116,72	116,59	116,98	117,63	116,76	-	-	-	-

Table 15: Results of grid search WAF. Each cell represents the percentage difference from the best total average fixed cost.

For the Mediterranean instance, the best temperature schedule is $[10^5 + 1, 0]$. The grid field results in the lowest average total fixed cost of $2,457,638$ USD from five runs.

	$\mathbf{10^{10}+1}$	$\mathbf{10^{9}+1}$	$\mathbf{10^{8}+1}$	$\mathbf{10^{7}+1}$	$\mathbf{10^{6}+1}$	$\mathbf{10^{5}+1}$	$\mathbf{10^{4}+1}$	$\mathbf{10^{3}+1}$	$\mathbf{10^{2}+1}$	$\mathbf{10^{1}+1}$
0	101,62	105,71	106,13	100,94	102,58	100,00	101,87	104,42	106,13	102,51
1000	103,40	105,71	106,13	100,94	102,58	100,06	101,87	104,42	-	-
10000	145,63	144,45	144,29	149,17	143,74	143,38	143,03	-	-	-
20000	147,59	147,15	146,49	143,61	145,75	141,86	-	-	-	-
50000	146,81	145,61	144,81	143,80	146,18	145,99	-	-	-	-

Table 16: Results of grid search Mediterranean. Each cell represents the percentage difference from best total average fixed cost.

7.2 Services for LS-MCFP

The construction of the service is performed in the Python coding language using relevant libraries. Python is utilized in the entire first stage of the two-stage algorithm.

The temperatures used to construct the final services in the SA metaheuristic are the optimal temperatures found in the grid search in section 7.1. The idea behind solving the LS-MCFP depends on creating more unique services than can feasibly exist due to the limited number of vessels available in each instance. A subset of these services will be selected when solving the LS-MCFP and optimizing the cargo flow. These unique services are created using the function *finalservice*$(temp0, tempend, tour)$ (see algorithm 12).

In *finalservice*$(temp0, tempend, tour)$, the capacity and the distance a vessel can travel is multiplied by a random scaler between $[0.5; 1.5]$. The SA metaheuristic forms new services by varying the capacity and distance a vessel can travel. The total amount of services created in the SA metaheuristic depends on the number of ports in the instance. Three scenarios will be created where the number of services designed for the LS-MCFP varies. The first scenario uses a minimum of 3 x *number of ports* services as input, while the second uses a minimum of 6 x *number of ports* services, and the last one a minimum of 9 x *number of ports* services. Each scenario is referred to as *low*, *mid*, and *high*, respectively. Note that these scenarios should not be mixed with the ones from LINER-LIB. Each unique service, found in the SA metaheuristic, in *bestserv* is added to the list *finalservice*. When the number of unique services in *finalservice* is $\geq$ *specified scalar* x *the number of unique ports*, the while loop breaks, and *finalservice* is returned. Note the services within the low scenario form a subset of those included in both the *mid*- and *high* scenarios, while the services within the *mid*-scenario constitute a subset of the services within the high scenario.

Algorithm 12 Create the services for the LS-MCFP

```
1:  procedure finalservice(temp0, tempend, tour)
2:      Finalservice = [ ]
3:      losninger = [ ]
4:      iteration = 0
5:      while length(Finalservice) ≤ length(tour) · scaler_i do        ▷ scaler_i ∈ [3,6,9]
6:          n = 5000
7:          cappercent = random(0.5, 1.5)
8:          distpercent = random(0.5, 1.5)
9:          vessel_capacity = vessel_capacity · cappercent
10:         vessel_dist = vessel_dist · distpercent
```

```
        Use simulated annealing 9                ▷ bestserv is output from SA algorithm
        for i in range(length(bestserv)) do
            if bestserv_i not in finalservice then
                append bestserv_i to finalservice
            end if
        end for
    end while
    return finalservice - see structure in table 9
end procedure
```

The performance of the algorithm in terms of computational demands can be found in table 17. The runs were executed with the following CPU and RAM configurations: AMD ryzen 5 7600X and 32 GB DDR5 RAM. The table shows the average time, in minutes, used for five runs of the SA metaheuristic for each of the three instances.

Data collection was overlooked during the construction of services in the following analysis of iterations per second for the SA metaheuristic. This oversight occurred during the initial construction phase, and the missing data were subsequently gathered in separate and independent runs using the same method employed in the grid search.

Instance	Run (iterations/sec)					Five runs avg.	Time (minutes)	High scenario (minutes)
	1	2	3	4	5			
Baltic	97.05	95.33	95.22	95.28	95.44	95.66	161.15	54.88
WAF	43.21	40.89	39.91	40.18	38.91	40.62	379.53	86.16
Mediterranean	28.74	26.58	26.64	26.82	26.79	27.11	568.59	101.40

Table 17: Table of computational times of the SA metaheuristic for each instance. Each run denotes the number of iterations per second. Five runs avg. is the average number of iterations per second across all five runs. Time (minutes) is the time it took to perform the grid search, and High scenario (minutes) is the time it took to design the services for the *high*-scenarios of the instances (based on the average ite/sec calculated).

7.3 Results of LINER-LIB instances

The LS-MCFP models are tested by implementing them in the programming language Julia (Julia n.d.) using the JuMP package, which is a domain-specific modeling language for mathematical optimization (JuMP n.d.). Furthermore, the optimization software HiGHS is used to solve the LS-MCFP models. The model for each instance will consider the three test scenarios, detailed in section 7.2, involving different numbers of services to evaluate and compare the model's performance under different circumstances.

Prior to running the models, the graphs $G(N, A)$ for all instances are constructed to consider the nodes and arcs of the generated services and the port modifications. The graphs will serve as the foundational network for the LS-MCFP models for which the services transport containers along paths. In addition, the shortest path algorithm is also performed on $G(N, A)$ in the subproblem. The number of nodes and arcs of the graphs are presented in table 18 along with the number of services.

In figure 18, $|N|$ is the number of nodes, including the ports visited by the services and the added port modification nodes from section 5. The notation $|A|$ denotes the number of arcs that establish connections between the ports of services and the modification nodes, encompassing origin/destination nodes and transshipment nodes. The number of transshipment arcs is captured in $|A_T|$, and $|S|$ denotes the number of services. For all three instances, the magnitude of $|A_T|$ dominates across the arcs. This is attributed to their quadratic growth expressed in equation 5.

Scenario	$G(N, A)$			Services
	$\|N\|$	$\|A\|$	$\|A_T\|$	$\|S\|$
		Baltic		
Low	189	3275	2758	38
Mid	337	11391	10430	74
High	531	27869	26326	111
		WAF		
Low	300	6814	2990	63
Mid	542	24668	23108	123
High	739	51899	49748	187
		Mediterranean		
Low	546	7172	5406	129
Mid	932	21682	18758	236
High	1318	44876	40794	352

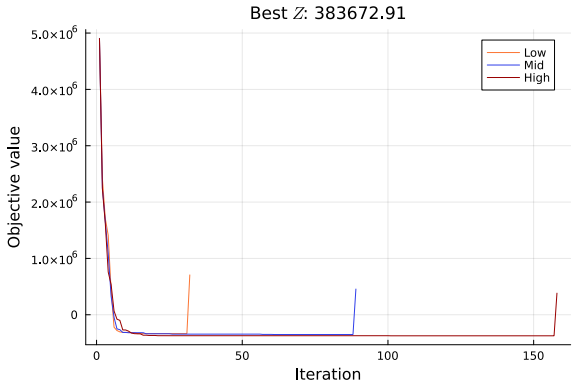
Table 18: Overview of $G(N, A)$ and services for each scenario within each instance. Note that $|A_T|$ is a subset of $|A|$.

Adding columns in the RMP and subsequently deriving corresponding solutions entails numerous model evaluations, each contributing to the final solution. The following assesses the solutions of the instances, including their intermediary solutions from the corresponding RMP. Regarding graph visualization, counting the number of services included in the solutions is difficult, as fractional services are allowed in the RMP. For example, if a subsolution only utilizes 0.1 of service y_s. To simplify the obscurity, ser-

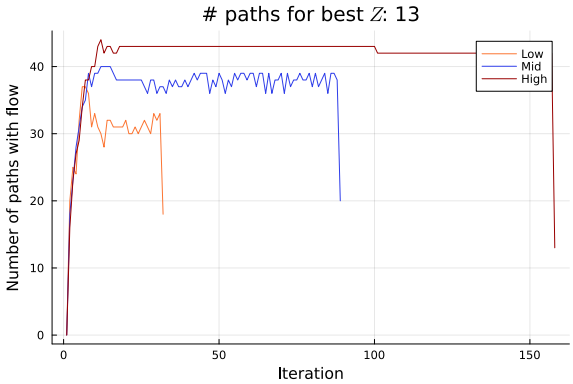
vices having a variable value larger than 0 are included in the plots of the convergence process.

7.3.1 Baltic

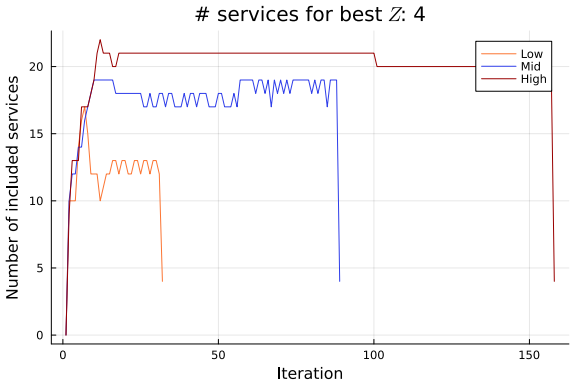
The first test instance is the Baltic Sea. It contains 12 ports and is modeled as a single-hub shipping network, as most commodities flow to or from DEBRV (Bremerhaven, Germany). The instance offers 4 x Feeder_450 and 2 x Feeder_800, each with a 450 and 800 FFE capacity, respectively. Figure 17 shows the progression in the objective value alongside the utilization of paths and services within the solutions, showcasing the improvement over iterations.



(a) Objective values.



(b) Number of paths in solutions.



(c) Number of services in solutions.

Figure 17: Performance of RMP and LS-MCFP on Baltic.

As previously mentioned, the initial solution of the RMP comprises solely penalty arcs. Consequently, the objective value commences at a higher point where neither paths nor services are included in the solution. However, once the column generation identifies paths with negative reduced costs, the model starts to converge fast, thus incorporating more profitable paths into the solution compared to the penalty arcs.

By looking at subfigure 17b and 17c, it becomes evident that after a few iterations, the scenarios start to find solutions where the number of services used and paths with container flow start to differ. Moreover, it is seen that the number of services- and paths used in the *low*- and *mid*-scenarios oscillate. This is due to the nature of column generation, as the number of paths added to the RMP varies from iteration to iteration, whereby the commodity flow changes. The same property applies to the services because the included services in the solution depend on the paths flowing profitable cargo. The *high*-scenario is subject to fewer improvements, which indicates that it obtains a good solution quite early in the process. Despite the fluctuations for the *low*- and *mid*-scenarios, the objective values seen in subfigure 17a converge similarly but result in different optima. This observation can be supported by the notion that while the three solutions include the same number of services in the final solution, the *high*-scenario employs fewer paths for container flow, ultimately leading to lower transportation costs entailing the best solution.

The best LS-MCFP model for Baltic obtains a total transportation cost of $3.837 \cdot 10^5$ USD, in which four services are responsible for sending commodities along 13 paths. The *low*- and *mid*-scenario obtains a cost of $7.074 \cdot 10^5$ USD and $4.567 \cdot 10^5$ USD respectively. The optimal solution of the corresponding RMPs attains objective values ranging from $-3.351 \cdot 10^5$ USD to $-3.737 \cdot 10^5$ USD, suggesting fairly similar profits. The significant spikes in the last iteration of the three subfigures signify the difference between the final objective values from the RMP and the LS-MCFP model. These spikes are caused by solving the LS-MCFP using the resulting paths from the last iteration of the RMP. Here, y_s is no longer relaxed in the LS-MCFP, thus acting as a binary variable. This forces the services to be either included in the solution or not, leading to the exclusion of numerous services due to the constraints imposed by the limited vessel fleet availability (see eq. 16 in section 6.2).

The similar objective values of the RMPs do not reflect similarities in the corresponding LS-MCFP solutions. This is evident by the solutions of the LS-MCFP, where fractional services are prohibited. Here, the objective values vary significantly, with the best solution being the *high*-scenario, which acquires costs of $3.837 \cdot 10^5$ USD. The worst solution is the *low*-scenario with a loss of $7.074 \cdot 10^5$. This is due to the RMP solutions utilizing different services despite attaining similar objective values. When y_s is tightened, these different services do not produce similar objective values in the LS-MCFP.

To visually illustrate the optimal solution, figure 18 shows the services alongside their corresponding flow for the *high*-scenario, offering a clear representation of the instance's dynamics.

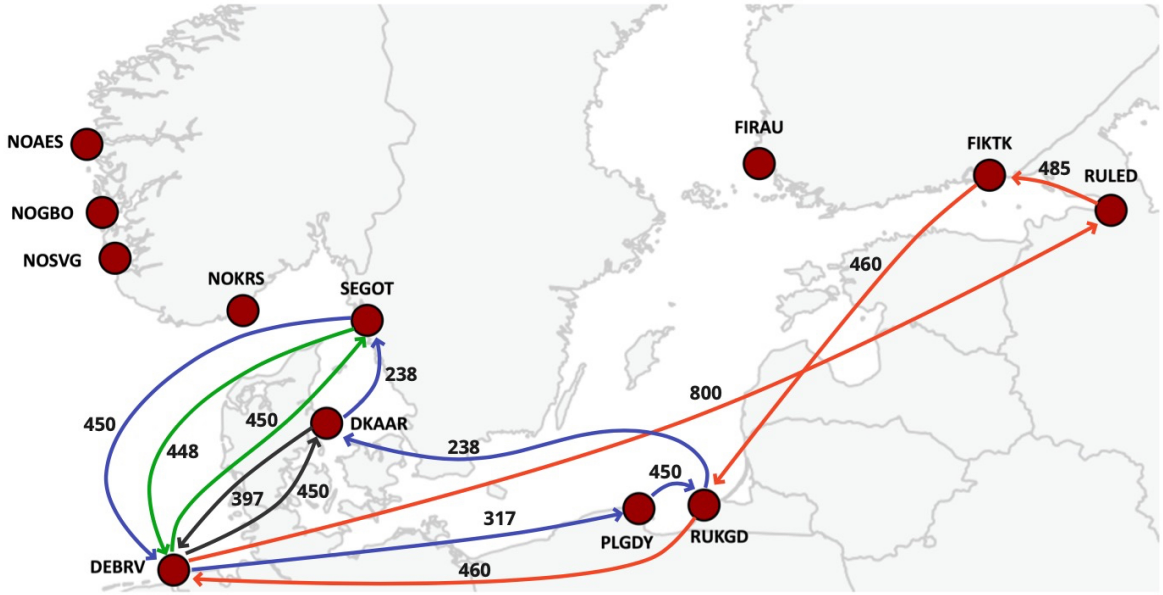
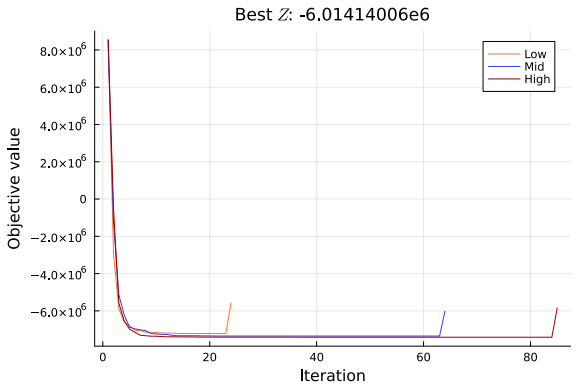


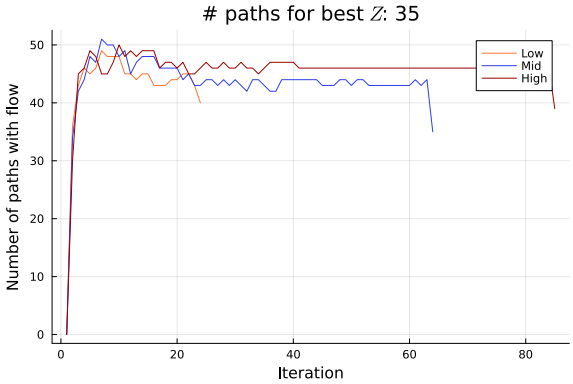
Figure 18: Resulting services of *high*-scenario Baltic.

7.3.2 WAF

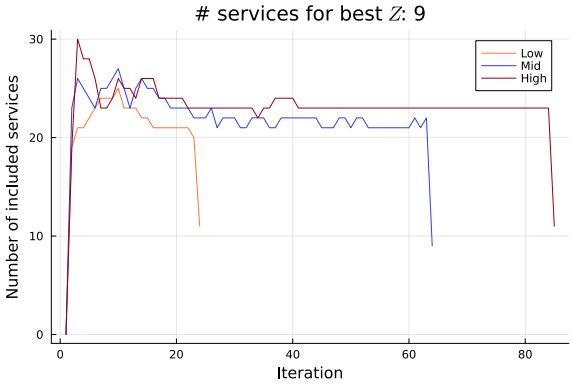
The next instance that will be applied for model testing is WAF, which contains 20 ports. ESALG (Algeciras, Spain) is the only port flowing more than 20 commodities in and out of its docks; hence, this instance is also modeled as a single-hub network. In WAF, the intricacy increases with additional commodities and a larger fleet, resulting in a more complex operational landscape. The fleet involves 14 x Feeder_450 and 28 x Feeder_800. The convergence of the models in WAF can be found in figure 19.



(a) Objective values.



(b) Number of paths in solutions.



(c) Number of services in solutions.

Figure 19: Performance of RMP and LS-MCFP on WAF.

The behavior of the RMP and its variables are somewhat similar to the results presented in Baltic. However, the solutions' fluctuations are notably more stable for this particular instance, suggesting fewer changes in the included services and the commodity flow.

Furthermore, the model shows consistent behavior across all three scenarios, evident in their close convergence as illustrated in subfigures 19b and 19c. This consistent behavior could stem from the subproblem identifying common paths across each model, contributing to their similarity in the number of paths and services. Although the convergence of the scenarios is similar, WAF exhibits that fewer services influencing the number of paths in the solution result in a better objective value.

All models demonstrate profitability with the *mid*-scenario emerging as the most successful scenario within WAF, resulting in a total cost of $-6.014 \cdot 10^6$ after solving the LS-MCFP, thus profits are acquired. The underlying RMP obtains an objective value of $-7.352 \cdot 10^6$ USD. The relatively small difference in the two objective values suggests that the effect of tightening y_s is somewhat negligible. This implies that several alternative paths have been generated for each commodity, which remains utilizable if certain services are excluded from the solution.

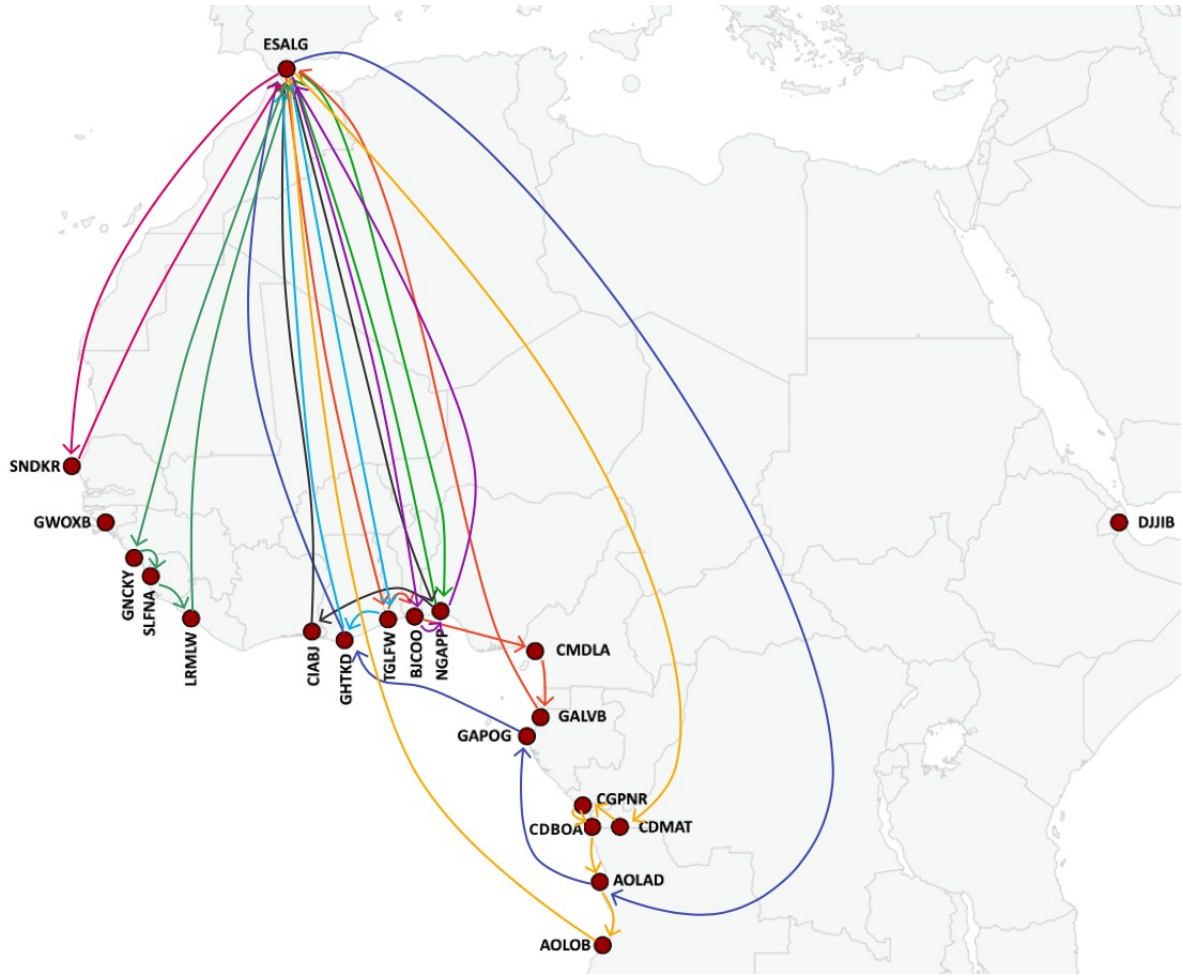


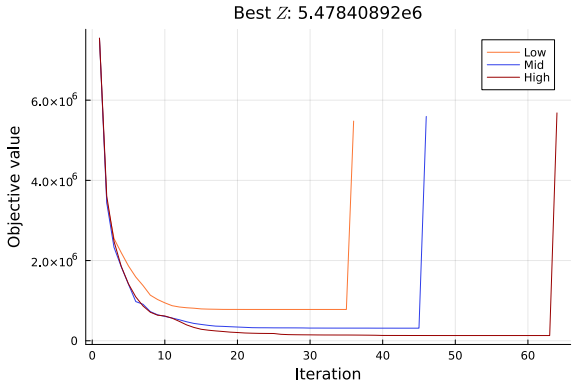
Figure 20: Resulting services of *mid*-scenario WAF.

Figure 20 shows the services of the *mid*-scenario in WAF. To prevent confusion, the numbers denoting the commodity flows are excluded. In addition, almost all ports but GWOXB (Bissau, Guinea-Bissau) and DJJIB (Djibouti, Djibouti) are visited. Hereby, a supply of 40 and a demand of 5 containers are disregarded in GWOXB, while 37 and 162, respectively, are neglected in DJJIB. The oversight of these ports can be attributed to low demand and supplies and long distances. The low demand at GWOXB suggests that the cost of visiting the port outweighs the potential revenue gain. DJJIB faces a similar situation, but its considerable distance results in even higher associated costs, further deterring its inclusion.

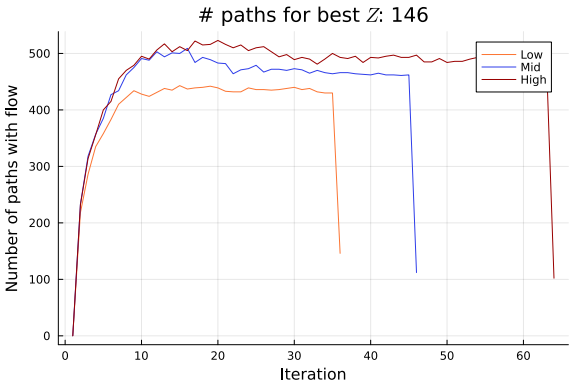
7.3.3 Mediterranean

Mediterranean is the last instance that has been selected for testing. The instance is modeled as a multi-hub network because 12 ports have more than 20 ingoing- and outgoing commodities. The intricacy of the model becomes more complex as the number of path- and service combinations increases. The fleet incorporates a new vessel class

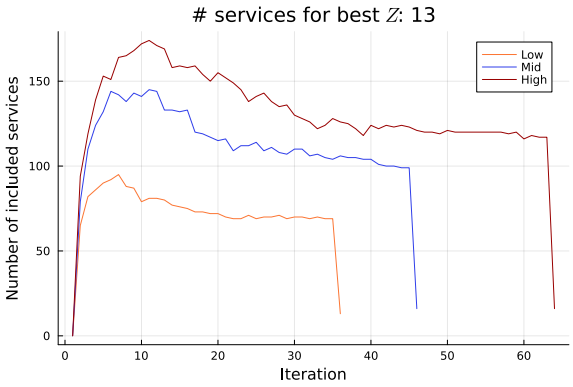
called Panamax with a corresponding capacity of 1200 FFE. The available fleet is 8 x Feeder_450, 8 x Feeder_800, and 4 x Panamax_1200. Figure 21 shows the convergence process of the models.



(a) Objective values



(b) Number of paths in solutions



(c) Number of services in solutions

Figure 21: Performance of RMP and LS-MCFP on Mediterranean.

The results indicate a notable trend where the convergence of the *mid-* and *high* scenarios occurs after fewer iterations than the Baltic and WAF instances, which are much simpler to solve. Conversely, the *low*-scenario exhibits a higher iteration count before convergence, which implies a lower variance in the number of iterations when the model is exposed to a varied number of available services. The *low*-scenario attains the best solution of $5.478 \cdot 10^6$ USD, which includes 13 services constituting 146 paths flowing 47.57% of the containers. When more services become available, the total cost increases, indicated by the *mid-* and *high*-scenario obtaining an objective value of $5.594 \cdot 10^6$ USD and $5.682 \cdot 10^6$ USD, with corresponding delivery rates of 45.19% and 44.18% respectively. By looking at subfigure 21c, the models exhibit a decrease in the number of services included in the solutions throughout the iterations of the RMPs. Despite this reduction, only small changes occur to the number of paths flowing containers when optimizing the RMP. In all three scenarios of the Mediterranean, the

solutions obtained in the LS-MCFP are significantly worse than those obtained in the RMPs. This happens because the constraints related to y_s are tightened, leading to a substantial decrease in the number of services with flow. As a result, the constraints are no longer perfectly fitted to the problem, causing a significant increase in costs.

Another interesting observation is that none of the RMPs manage to obtain a negative objective value, signifying an absence in profits. This issue impacts the LS-MCFP, making it impossible to generate profits.

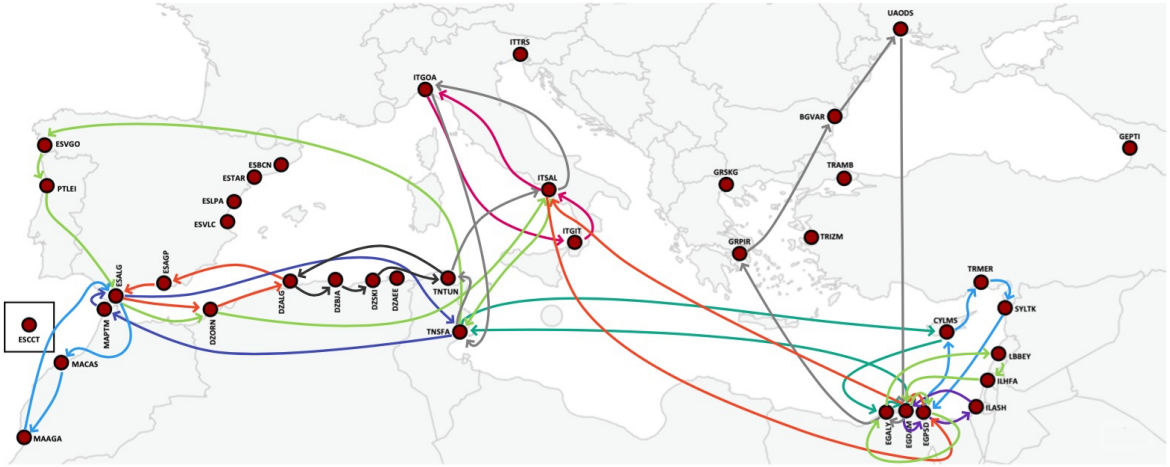


Figure 22: Resulting services of *low*-scenario Mediterranean.

The included services in the LS-MCFP model obtained in the *low*-scenario can be found in figure 22. Note that the commodity flows are excluded to simplify the illustration.

7.3.4 Summary of results

Table 19 shows the solutions of the three scenarios applied to the Baltic, WAF, and Mediterranean instances. The objective value of the LS-MCFP solution, denoted as Z, quantifies the total transportation cost of the solution in USD. Z_{RMP} is the objective value of the last iteration of the RMP, which similarly signifies the total transportation cost of the solution in USD. As the objective is to minimize costs, a negative value of Z and Z_{RMP} indicates a profit surplus. Total f^C is the total revenue calculated by the sum of the delivered commodities multiplied by the corresponding revenue of successfully delivering a single FFE in USD. Moreover, the transported cargo (%) signifies the delivery rate of all containers, where the remainder constitutes the share of forfeited containers. The number of paths flowing containers is denoted by $x_k^p > 0$, and the number of services included in the final is represented as $y_s = 1$. LS-MCFP (minutes is the computational time of solving the LS-MCFP with binary y_s, and RMP (minutes) is the computational time of solving the subproblem.

Scenario	Z	Z_{RMP}	Total f^C	Transported cargo (%)	Total $x_k^p > 0$	Total $y_s = 1$	LS-MCFP (minutes)	RMP (minutes)
				Baltic				
Low	$7.074 \cdot 10^5$	$-3.351 \cdot 10^5$	$30.779 \cdot 10^5$	71.61%	18	4	0.278	0.350
Mid	$4.567 \cdot 10^5$	$-3.510 \cdot 10^5$	$32.067 \cdot 10^5$	75.33%	20	4	0.825	1.160
High	$\mathbf{3.837 \cdot 10^5}$	$\mathbf{-3.737 \cdot 10^5}$	$\mathbf{31.773 \cdot 10^5}$	**76.91%**	**13**	**4**	**3.903**	5.576
				WAF				
Low	$-5.560 \cdot 10^6$	$-7.217 \cdot 10^6$	$14.863 \cdot 10^6$	99.20%	40	11	0.341	0.390
Mid	$\mathbf{-6.014 \cdot 10^6}$	$\mathbf{-7.352 \cdot 10^6}$	$\mathbf{14.387 \cdot 10^6}$	**95.21%**	**35**	**9**	**2.113**	2.596
High	$-5.845 \cdot 10^6$	$-7.415 \cdot 10^6$	$14.825 \cdot 10^6$	98.84%	39	11	9.667	7.340
				Mediterranean				
Low	$\mathbf{5.478 \cdot 10^6}$	$\mathbf{0.781 \cdot 10^6}$	$\mathbf{2.362 \cdot 10^6}$	**47.57%**	**146**	**13**	**42.117**	6.690
Mid	$5.594 \cdot 10^6$	$0.314 \cdot 10^6$	$2.372 \cdot 10^6$	45.19%	112	16	178.334	27.880
High	$5.682 \cdot 10^6$	$0.130 \cdot 10^6$	$2.271 \cdot 10^6$	44.18%	102	16	1068.253	77.201

Table 19: Solutions and performance metrics of the three scenarios within the Baltic, WAF, and Mediterranean instances. The best solutions are highlighted in bold. All models were run with a 1.8 GHz Dual-Core Intel Core i5 processor and 8 GB memory.

The metrics indicate that the LS-MCFP method results in diverse solutions for the three instances. Baltic and WAF are modeled as single-hub networks where the resulting flow entails profits in WAF while losses are incurred in Baltic. The relationship between Baltic's objective value and the number of services available in each scenario highlights a trend that greater availability of services implies reduced losses. Interestingly, this relationship does not apply to WAF nor Mediterranean as the *mid*- and *low*-scenario, for the two instances, respectively, have the lowest objective value. WAF shows promising results in all scenarios as significant profits are made. This confirms that the generated services from the hub-based metaheuristic perform well for this particular instance since the entire solution space does not have to be explored to find a good solution that entails profits.

The solutions of the scenarios in Baltic all utilize 4 services. Despite equivalent utilization, the scenarios result in different objective values, including delivery rates from 71% to 77%. Regarding the *high*-scenario, which obtains the best objective value, the relatively low delivery is due to the neglect of Norway and FIRAU. This neglection can be attributed to a mixture of the vessel capacities and services requiring multiple vessels, depicted in figure 18.

Another point worth highlighting for WAF is the overall vessel utilization, which, across all three scenarios, yields a cargo delivery rate exceeding 95%. While achieving more than 95% delivery rate generates increased revenue from container delivery, it also leads to higher costs, as shown in the *low*- and *high*-scenarios. These costs do not break even with the revenue generated from delivering the last 2-5% of the containers.

The high delivery rates are caused by the large fleet available, allowing for operating an extensive number of services to accommodate almost all demands.

Mediterranean incurs losses for all three scenarios, similar to Baltic. The lowest cost is obtained in the *low*-scenario where 146 paths are flowing containers utilizing 13 services. The complexity of Mediterranean is reflected in the results of the LS-MCFP as the models yield unsatisfactory outcomes, with each scenario incurring losses exceeding 5 million USD. This involves a delivery rate averaging 45.65%, signifying that over 50% of all containers are forfeited.

The computational time spent on running the models scales proportionally with the number of commodities involved across the three instances and the number of services. Here, Baltic spends a maximum of 5 minutes managing 22 commodities with 38, 74, and 111 services available, whereas WAF uses a maximum of 10 minutes to handle 37 commodities across 63, 123, and 187 services. The complexity in Mediterranean influences the performance as the computational time for solving the models ranges from 48-1173 minutes for managing 365 commodities with 129, 236 and 352 available services.

The complexities of solving the Mediterranean instance are further revealed when comparing the results to the literature presented in section 2. Particularly, Karsten et al. (2017) and Koza, Desaulniers, and Røpke (2020) obtain high transportation costs of $2.42 \cdot 106$ USD and $2.44 \cdot 10^6$ USD, respectively; however, their findings are twice as good as the results presented in this thesis, and they obtain delivery rates of above 75%. Their results exceed the findings presented in this thesis and solve a more complex problem, capturing shipping dynamics with greater accuracy. Considering the Brouer et al. (2014) paper as well, all three papers gain profits in the Baltic and WAF instances. However, the profit of $6.014 \cdot 10^6$ USD obtained within WAF using this method surpasses the results of the presented papers. Note that only direct comparisons can be made to Brouer et al. (2014), as this is the only paper of the three that considers the same problem.

8 Discussion

The task of solving the SND problem and LS-MCFP with column generation is not an easy endeavor. Given the complexity of the task, certain aspects and methodologies employed could benefit from further enhancements. These possible improvements will be discussed in the following section. Several topics discussed below have not been considered or handled due to the time limitations imposed by the thesis deadline.

8.1 Limitation of scope

The LINER-LIB benchmark suite serves the purpose of testing and improving upon real-world LSNDP solution methods. The dataset offers a platform to delve into diverse scenarios encompassing multiple variables whereby trade-offs and complexities encountered in real-world scenarios can be simulated when addressing the LSNDP. The scope of this thesis was set to consider Baltic, WAF, and Mediterranean. In regards to the variables that are taken into account, the following ones remain unutilized or unincorporated.

The first variable is the time-constraints that inherently are part of any transportation system. Customers' expectations consistently lean towards fast order fulfillment. The implication of leaving out the time constraints results in a reduced representation of real-world scenarios.

Secondly, the variables not considered are the variable vessel speeds of v^F. By fixing the vessels' speed, the complexity of optimizing the services created is lessened. However, this comes at the cost of overestimating the required number of vessels sailing the services. As mentioned in section 4.1, an additional vessel is added to the service as soon as the already allocated vessels cannot travel the total distance of the service at s^F_*. This results in unnecessarily high fixed costs of services. Further, it limits the possibility of another service being used in the LS-MCFP, thereby minimizing the potential amount of cargo being delivered. The thesis omits information about vessel drafts and the permitted drafts in ports. Consequently, certain vessels might visit ports where the vessel's draft is too large for the given port.

8.2 Design choices of the network

Designing the logic behind the SA metaheuristic to create the services is not an easy task. The intricate nature of service design has led to specific design choices that simplify the implementation of vessel capacity and travel distance. Further, due to how capacity is implemented, the supply of the ports has been simplified to consist of an average of all market orders that the ports are a part of. These simplifications were made such that neither demand nor supply were disregarded in the SND. The capacity of vessels in the SND problem is implemented such that the ships travel to ports, load the cargo, and once all the capacity is fully utilized, the vessel ends its service. This leads to potentially underestimating the amount of ports a vessel can visit regarding capacity. Alternatively, a vessel could transport cargo from an earlier port in the service, which could then be unloaded at a subsequent port. This strategy allows for better capacity utilization by freeing up capacity and not cutting a service short regarding capacity.

8.3 Implementation of hubs

Further, how hubs are implemented by including hubs in every service results in low fixed costs for services in the Baltic and Mediterranean instances, compared to WAF in the grid search in section 7.1. The only hub in WAF is ESALG. Because of ESALG's geographic location, the structure and geographical positioning of the other ports within the WAF instance, and the requirement that all services must contain at least one hub port, WAF services tend to be long, requiring a significant number of vessels. This happens because vessel allocation for services is based on the total distance of the service, and hubs are only added after the initial services are constructed. Therefore, if the initial service is situated far from the hub and the hub is subsequently added to the service, the distance of the service is large. One could try to model WAF as a conveyor network rather than a hub-centered network (see figure 4 in section 4). This would result in shorter services but more transshipments, thus creating a trade-off that needs to be managed. Moreover, alternative methods for identifying hubs could be investigated. As outlined in section 4, this thesis defines hubs as ports involved in 20 or more market orders. Alternatively, ports with fewer market orders but strategically advantageous geographic locations could have been designated as hubs. In this scenario, these ports would serve more as distribution centers, prioritizing optimal geographic positioning over minimizing necessary transshipments for fulfilling market orders.

8.4 Connecting services for transshipment

The way transshipment facilitation is implemented in the SND problem could be improved. As mentioned in section 4.1, Kruskal's algorithm is applied to create an MST based on the average coordinates of services and then generate the transshipment facilitation based on the MST. Instead of using the average coordinates between services, an all-to-all-to-all distance of the ports closest to each other in the services could have been used. This implies that if a service consists of a group of closely located ports and a lone port situated farther away from the cluster, and if the lone port has a neighboring port close by, it might be more advantageous to construct the transshipment node based on the lone port and its neighboring port.

In continuation, the number of ports allowing for transshipment in a service has not been explored. The method used to implement transshipment facilitation only ensures that at least one transshipment is possible in every service. It could be the facilitation of more transshipments in the services to improve the amount of cargo delivered, as there might be more feasible and cheaper paths for transporting the cargo.

8.5 Hyperparameter tuning for SA metaheuristic

As mentioned earlier, the temperature schedule of the SA metaheuristic is essential for controlling the balance between exploration and exploitation during the extrema search. This leads to the importance of efficiently finding the best temperature schedule for the particular setting of the SA metaheuristic. The decision to employ grid search for temperature optimization was rooted in its straightforward and systematic methodology, offering convenience when implementing. However, this approach does come with downsides. The drawback of grid search for temperature schedule optimization lies in its computational demand, resulting in significant time consumption. For Mediterranean, on average, 27.11 iterations per second were calculated for the SA metaheuristic, indicating that the initial grid search took about 9.47 hours, as it required $925,000$ iterations. Further, the exploration of the grid search is limited to the predefined temperature schedules, meaning that if the optimal schedule falls outside these predefined parameters, there exists no chance of discovering it within this methodology. In addition, the temperatures essentially are continuous parameters, meaning that a discrete grid search will most likely lead to missing the optimal temperature schedule and a loss of performance.

The obvious solution to this problem would be to look at other optimization methods. One that would be interesting to implement would be Bayesian optimization for hyperparameter tuning. In short, Bayesian optimization chooses the next set of parameters based on a probabilistic model of improvement to the objective function. It balances exploration and exploitation while minimizing the number of evaluations by efficiently navigating the parameter space (Bergamin and Madsen n.d.).

Further, fitting the temperature schedule to each instance could potentially overfit the SA metaheuristic to the instances. The choice of fitting instance-wise was made due to the diverse characteristics of each of the instances. It was determined that the temperature schedule needed to be fitted to each instance to obtain the best possible results in the instances. The diversity of the instances also calls for differentiation regarding the exploration and exploitation of the solutions generated in the SA metaheuristic. The Baltic instance with a smaller solution space seems to benefit from more exploration. In contrast, the WAF and the Mediterranean instances with larger solution spaces might benefit more from finding a "good" solution and then trying to better that solution by investigating the neighboring solutions.

8.6 Handling binary variables

The binary variables y_s linked to the services significantly influence the determination of the optimal solution in the LS-MCFP. This arises from the significant difference

between the solutions obtained in the final iteration of the RMP and the final solution of the LS-MCFP. The discrepancy becomes evident due to the LS-MCFP being solved with tightened variables in terms of y_s. This is particularly applicable when numerous fractional services are included in the subproblem solutions, as the y_s no longer compete with inclusion or exclusion due to their fractional nature.

Ideally, the objective is to minimize these discrepancies to prevent LS-MCFP from obtaining a solution far from the subproblem's solution. Consequently, the current approach can lead to solving a different problem than intended because the focus should be on setting up the problem to find the most fitting solution that adheres strictly to the binary constraints. An effective strategy to tackle this could involve leveraging the services included in the final solution. For instance, one approach could entail deploying a heuristic consisting of an iterative process where, in each iteration, a small change is made in the construction of the services, whereafter the LS-MCFP is solved. This process continues until a certain termination criteria is met. The method can utilize a heuristic's properties to potentially converge towards an optimal solution. Moreover, it leverages a feedback mechanism embedded within the mathematical model, assessing whether the solution shows improvement based on the small change in the services. This iterative process enables adjustments based on this feedback, contributing to the pursuit of an enhanced solution.

Another more straightforward approach would be to utilize the resulting services of the LS-MCFP in the RMP. Hereby, a new constraint could be introduced where these services equal 1 in the RMP. This enables the column generation to find more alternative paths that might improve the solution, as paths that do not utilize the resulting services are disregarded.

8.7 Performance of LS-MCFP

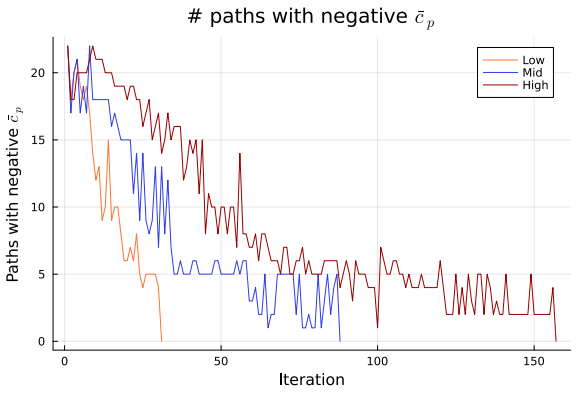
The dissimilarity in the nature of the three instances and the obtained results imply that the instances require a more tailored service modeling within the SND problem's hub-modeling framework, suggesting that a singular strategy might not effectively encapsulate the intricacies of these diverse instances.

Examining the Baltic instance, including the obtained services in the optimal solution of the LS-MCFP shown in figure 18, the Norwegian ports and FIRAU (Rauma, Finland) are disregarded. This is because the existing fleet is fully deployed, thereby constraining any possibility of further expansion. This results in the corresponding containers being forfeited as a result. To address this issue, one could improve the vessel utilization in terms of vessel speeds described in 8.1. Regarding the presented findings, the demand and supply of the unvisited ports in Norway constitute 98 containers. It

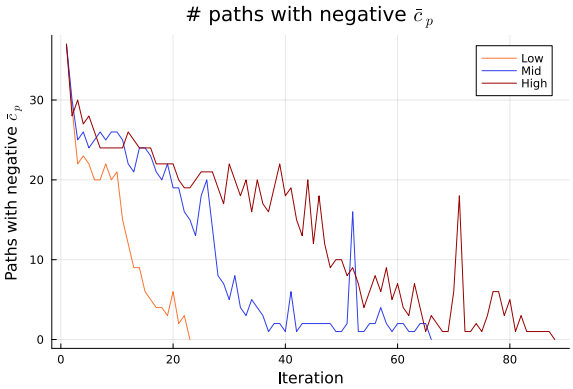
is, therefore, important to ensure that the revenue generated from these container deliveries surpasses the fixed costs associated with establishing a new dedicated service between DEBRV and Norway to reduce potential losses incurred in the process.

The WAF instance could also benefit from vessel speed, as it faces a comparable issue regarding balancing revenue against total costs. Despite having a more extensive fleet in WAF, which leads to a higher delivery rate across all three scenarios (see table 7.3), the model interestingly identifies the *mid*-scenario, with the lowest delivery rate of 95.21% among the three scenarios, as the optimal solution. This counterintuitive finding suggests that although higher delivery rates yield more revenue, it is necessarily not beneficial to deliver the last 4.79% containers in terms of costs. This phenomenon could be explained by the notion that the potential revenues gained from the disregarded ports cannot break even with the fixed- and variable costs. Consequently, it might be beneficial to enhance the performance of the employed services in the *mid*-scenario by considering variable vessel speeds.

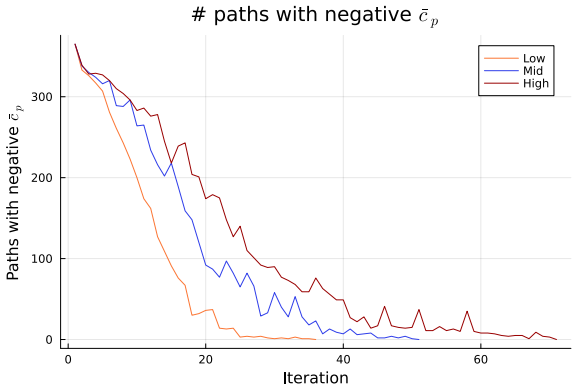
As mentioned earlier, the Mediterranean instance involves a high complexity level, leading to intricate services and paths. The depth of this complexity is evident in the results shown in table 7.3, where every scenario demonstrates a delivery rate below 50%, culminating in substantial financial losses. This correlation could be attributed to the relatively small fleet size compared to the large volume of commodities that require delivery. This characteristic is in contrast to the WAF instance. As a result of the long services in Mediterranean, ports are disregarded because multiple vessels are allocated on the same services. Here, speed optimization, also discussed for the Baltic and WAF instances, is a viable solution for tackling this issue. By employing this method, it becomes possible to optimize vessel allocation for a specific service, thereby releasing these resources for alternative usage. Further, the complexity of the Mediterranean instance influences the services within the solution. This leads to intricate paths that visit the same ports, implying increased costs due to the utilization of transshipment in these paths, incurring additional expenses.



(a) Baltic



(b) WAF



(c) Mediterranean

Figure 23: Paths with negative reduced cost in RMP.

Another interesting observation is the counterintuitive aspect of the number of iterations required for the LS-MCFP model to reach the optimum solution across the three instances. From figure 17, 19 and 21 shown in section 7.3, it is noticed that the Baltic instance uses more iterations before it fully converges compared to WAF and Mediterranean. This observation challenges the expected relationship between complexity and model performance. Typically, larger and more complex models are anticipated to face greater difficulty achieving an optimal solution. However, Baltic presents a contradiction to that notion. To get an idea of why this happens, figure 23 illustrates the number of paths with negative reduced costs $\bar{c}_p$ that are added to the RMP in each iteration for the instances. The challenge in obtaining an optimal solution in the RMP for Baltic is evident for the *mid*- and high scenario. This is because the number of paths with negative $\bar{c}_p$ fluctuates across several iterations, where only few paths are added to basis, before it finds a solution that meets the optimality criterion.

9 Conclusion

This thesis shows how one could implement a simple service-first and flow-second algorithm for solving the LSNDP.

The service-first stage was solved using the SA metaheuristic, which allowed us to explore the solution space for services, favoring minimized total fixed costs. The SA metaheuristic, featuring hill-climbing capabilities, enables the search for a global minimum in terms of the fixed costs of the services while trying to avoid local minima. The SA metaheuristic was used to construct three sets of services constituting a different number of generated services. These sets contributed to a sensitivity analysis of the results of the LS-MCFP. The objective of the flow-second stage was to solve the LS-MCFP by minimizing both the fixed- and variable costs when flowing cargo. Dantzig-Wolfe decomposition was utilized to decompose the mathematical structure of the LS-MCFP, thus reducing the size of the flow model.

The results showed that the hub-centered network design does not apply equally well across the instances, suggesting the need for a more tailored approach. Here, WAF showed promising profits where the hub-centered network design and large fleet implied high delivery rates. The LS-MCFP struggled with low delivery rates in the Baltic and Mediterranean instances due to smaller fleets, suggesting that there is still room for improvement in the logic behind the SA metaheuristic for constructing the services. In particular, variable vessel speed would be intriguing to implement as it allows for lower bunker consumption, thereby lowering fixed costs and better utilization of the available fleets in the instances.

A correlation emerged between the number of available services and the complexity of the instance. This correlation affects the model, leading to the inclusion of paths in the RMP that might not necessarily be the most beneficial for enhancing the solutions of the LS-MCFP. This arises from the models' perception that they have identified cost-efficient paths for the commodities. When y_s is set binary, these cheaper paths using fractional services are not beneficial for the final solutions of the LS-MCFP.

Concluding on the thesis, we have shown that the implemented two-stage algorithm of service-first and flow-second for solving the LSNDP can be used as the foundation for further development to improve the achieved results and further reflect real-world complexities and issues regarding liner shipping planning.

10 Future work

Further developing and improving the proposed method entails leveraging the unutilized variables outlined in section 8.1. This would ultimately result in a problem setting that

reflects real-world liner shipping industry dynamics more accurately. Another intriguing facet includes looking into vessel speed optimization. Implementing variable vessel speeds could reduce the fixed costs of services as the current method for calculating vessel deployment overestimates the required number of vessels.

Further exploration into optimizing hyperparameters within the SA metaheuristic could also be interesting to improve upon. Given the continuous nature of the temperature variable, employing Bayesian optimization for hyperparameter tuning might reduce the fixed costs of the generated services. This optimization could potentially enhance overall results. Regarding the LS-MCFP, the total transportation cost heavily depends on the solution's services. Addressing the challenge of high costs could involve implementing an improvement heuristic discussed in section 8.6. Thereby, the solution of the LS-MCFP is leveraged as a feedback mechanism, allowing iterative improvement of the services provided to obtain a more optimized and cost-effective solution.

Lastly, broadening the scope of instances for the implemented two-stage algorithm would be intriguing. Each new instance offers valuable insights into the performance of the proposed method and areas for enhancement. While the process is manageable, the primary challenge lies in the time investment required to run the two-stage algorithm.

11 Learning outcome and project plan

To overcome the extent of the thesis, a project plan was initially made, providing an overview of the subprojects and milestones. The project plan and its description can be found in the appendix. During the course of the project, there have been insignificant changes to the project plan, entailing a somewhat smooth progress. However, the advanced aspects of the metaheuristic and the large-scale modeling have caused concerns as it was challenging to mathematically realize the high level of abstraction. This led us to reevaluate the scope where a different approach was considered that solely involved the Baltic and WAF instances, but was quickly discarded. After properly defining the scope, we decided to split up so each was responsible for one stage within the two-stage algorithm. This approach not only reduced the time spent on implementing the two stages but also opened up a new opportunity to work on the service scenarios, an aspect we had not initially factored in. This division furthermore enabled us to critically evaluate each other's work, both in terms of code implementation and written content. Overall, the project work has been characterized by trust. We established a framework built on mutual trust, allowing us to manage with relatively loose internal deadlines. This approach functioned smoothly because we shared a collective understanding that we were equally responsible for delivering applicable content. This formed the backbone of our project work, fostering a sense of reliability and accountability among us.

Throughout this project, we successfully achieved several predetermined learning objectives. The objectives encompass skills and knowledge within project management, mathematical modeling, and containerized liner shipping. In terms of project management, we initially established a project plan accompanied by a detailed project description. We continuously revisited this plan throughout the process to guarantee precise resource allocation to each subproject. This approach ensured that our resources were strategically allocated, preventing excessive time allocation to subprojects with lower priority. As a result, we maintained a focused and efficient workflow, investing our time and effort where it was most impactful and crucial to the project's success. Regarding the proposed metaheuristic and MIP formulation, we have gained a better understanding of metaheuristics, their concepts, and how to construct large-scale mathematical models. To conduct a critical assessment and analyze the performance of the proposed method, the models were tested on relevant instances from the LINER-LIB benchmark suite. This included a comparative examination with findings from various literature that proposed other solutions for addressing the LSNDP.

References

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). "Network Flows: Theory, Algorithms and Applications". In: *Prentice Hall, Englewood Cliffs, NJ*, chapter 17.

Álvarez, J. F. (2009). "Joint routing and deployment of a fleet of container vessels". In: *Maritime Economics and Logistics*, pp. 186–208.

Bergamin, Federico and Kristoffer H. Madsen (n.d.). "Bayesian Optimization". In: *Unknown* ().

Brouer, B. D. et al. (2014). "A Base Integer Programming Model and Benchmark Suite for Liner-Shipping Network Design". In: *Transportation Science*, pp. 281–312.

Christensen, M. et al. (2021). "Liner Shipping Network Design". In: *Network Design with Applications to Transportation and Logistics*, chapter 15.

comission, European (n.d.). *About UN/LOCODE Codes for Ports and other Locations*. URL: https://joinup.ec.europa.eu/collection/uncefact/solution/unlocode-codes-ports-and-other-locations/about.

Crainic, T. G. and M. Gendreau (2021). "Heuristics and Metaheuristics for Fixed-Charge Network Design". In: *Network Design with Applications to Transportation and Logistics*, chapter 4.

Dai, W., X. Sun, and S. Wandelt (2016). *Finding Feasible Solutions for Multi-Commodity Flow Problems*. National Key Laboratory of CNS/ATM.

Delahaye, D., S. Chaimatanan, and M. Mongeau (2018). *Simulated Annealing: From Basics to Applications*. HAL open science.

Economic Co-operation, Organisation for and Development (n.d.). *Ocean shipping and shipbuilding*. URL: https://www.oecd.org/ocean/topics/ocean-shipping/.

Gilmore, P. C. and R. E. Gomory (1961). "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research*, pp. 849–859.

Julia (n.d.). *The Julia Programming Language*. URL: https://julialang.org.

JuMP (n.d.). *JuMP*. URL: https://jump.dev/JuMP.jl/stable/.

Karsten, C. V. et al. (2017). "Time constrained liner shipping network design". In: *Transportation Research*, pp. 152–162.

Koza, D. F., G. Desaulniers, and S. Røpke (2020). "Integrated Liner Shipping Network Design and Scheduling". In: *Transportation Science*, pp. 512–533.

Maersk (n.d.). *What do FEU and FFE mean?* URL: https://www.maersk.com/support/faqs/feu-and-ffe-mean.

programiz (n.d.). *Kruskal's Algorithm*. URL: https://www.programiz.com/dsa/kruskal-algorithm.

Sakarovitch, M. (1983). *Complements on Duality: Economic Interpretation of Dual Variables*. Springer, pp. 142–155.

ShipsGo (n.d.). *What are Hub Ports?* URL: https://blog.shipsgo.com/what-is-hub-ports.

Teodor, G. C. and M. Hewitt (2021). "Service Network Design". In: *Network Design with Applications to Transportation and Logistics*, chapter 12.

Appendix

Learning objectives

- Describe and explain how the shipping companies operate and how a liner shipping network design can help them optimize their supply chain.
- Use metaheuristics to generate fixed services for the LSNDP.
- Formulate a mathematical model to model operational difficulties and constraints within maritime transportation as a MCFP.
- Test model on relevant instances from the LINER-LIB dataset.
- Conduct a critical assessment and analyze the performance of the experimental results.
- Apply and describe basic economic theory and concepts related to the shipping industry.
- Gain a better understanding of what it means to write an academic report.
- Be critical of each other's work.
- Resolve internal conflicts in a constructive manner.
- Learn to use techniques for problem-solving in the group.
- Improve competencies and abilities within project management and project planning.

Project plan

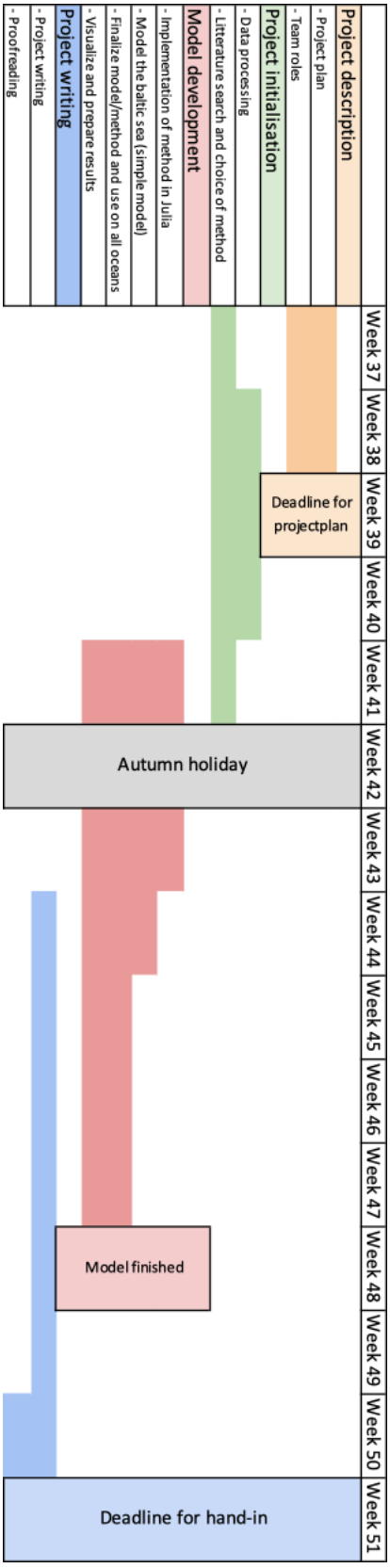


Figure 24: Gantt-chart showing subprojects

Baltic paths

Table 20: Baltic paths

Path	Total flow in FFE
SEGOTs → SEGOT86 → DEBRV86 → DEBRVt	448
DEBRVs → DEBRV86 → SEGOT86 → SEGOTt	450
FIKTKs → FIKTK13 → RUKGD13 → DEBRV13 → DEBRVt	162
RUKGDs → RUKGD76 → DKAAR76 → SEGOT76 → DEBRV76 → DEBRVt	7
PLGDYs → PLGDY76 → RUKGD76 → DKAAR76 → SEGOT76 → DEBRV76 → DEBRVt	231
SEGOTs → SEGOT76 → DEBRV76 → DEBRVt	212
DEBRVs → DEBRV13 → RULED13 → FIKTK13 → FIKTKt	187
DEBRVs → DEBRV13 → RULED13 → RULEDt	613
DEBRVs → DEBRV1 → DKAAR1 → DKAARt	450
DEBRVs → DEBRV76 → PLGDY76 → PLGDYt	98
DKAARs → DKAAR1 → DEBRV1 → DEBRVt	397
RULEDs → RULED13 → FIKTK13 → RUKGD13 → DEBRV13 → DEBRVt	298
DEBRVs → DEBRV76 → PLGDY76 → RUKGD76 → RUKGDt	219

WAF paths

Table 21: WAF paths

Path	Total flow in FFE
GHTKDs → GHTKD1 → ESALG1 → ESALGt	227
GALBVs → GALBV9 → ESALG9 → ESALGt	54
ESALGs → ESALG1 → AOLAD1 → GAPOG1 → GAPOGt	2
ESALGs → ESALG23 → SNDKR23 → SNDKRt	565

Continued on next page

Table 21: WAF paths (Continued)

Path	Total flow in FFE
SNDKRs → SNDKR23 → ESALG23 → ESALGt	189
CIABJs → CIABJ15 → ESALG15 → ESALGt	326
ESALGs → ESALG39 → CDMAT39 → CGPNR39 → CDBOA39 → CDBOAt	60
NGAPPs → NGAPP5 → ESALG5 → ESALGt	88
ESALGs → ESALG1 → AOLAD1 → AOLADt	696
ESALGs → ESALG5 → NGAPP5 → NGAPPt	800
ESALGs → ESALG1 → AOLAD1 → GAPOG1 → GHTKD1 → GHTKDt	102
CGPNRs → CGPNR39 → CDBOA39 → AOLAD39 → AOLOB39 → ESALG39 → ESALGt	19
AOLOBs → AOLOB39 → ESALG39 → ESALGt	51
GAPOGs → GAPOG1 → GHTKD1 → ESALG1 → ESALGt	8
BJCOOs → BJCOO79 → NGAPP79 → NGAPP5 → ESALG5 → ESALGt	94
ESALGs → ESALG39 → CDMAT39 → CGPNR39 → CGPNRt	133
ESALGs → ESALG39 → CDMAT39 → CDMATt] 220.0; Any[ESALGs → ESALG9 → TGLFW9 → TGLFWt	353
ESALGs → ESALG17 → TGLFW17 → GHTKD17 → GHTKDt	613
ESALGs → ESALG105 → GNCKY105 → GNCKYt	267
ESALGs → ESALG15 → NGAPP15 → CIABJ15 → CIABJt	317
ESALGs → ESALG15 → NGAPP15 → NGAPPt	483
ESALGs → ESALG79 → BJCOO79 → BJCOOt	510
GNCKYs → GNCKY105 → SLFNA105 → LRMLW105 → ESALG105 → ESALGt	59
ESALGs → ESALG9 → TGLFW9 → BJCOO9 → CMDLA9 → CMDLAt	274

Continued on next page

Table 21: WAF paths (Continued)

Path	Total flow in FFE
ESALGs → ESALG105 → GNCKY105 → SLFNA105 → SLFNAt	41
ESALGs → ESALG105 → GNCKY105 → SLFNA105 → LRMLW105 → LRMLWt	142
ESALGs → ESALG17 → TGLFW17 → TGLFW9 → BJCOO9 → CMDLA9 → GALBV9 → GALBVt	128
CMDLAs → CMDLA9 → GALBV9 → ESALG9 → ESALGt	286
TGLFWs → TGLFW9 → BJCOO9 → CMDLA9 → GALBV9 → ESALG9 → ESALGt	140
ESALGs → ESALG17 → TGLFW17 → TGLFWt	59
ESALGs → ESALG79 → BJCOO79 → NGAPP79 → NGAPPt	290
LRMLWs → LRMLW105 → ESALG105 → ESALGt	52
ESALGs → ESALG39 → CDMAT39 → CGPNR39 → CDBOA39 → AOLAD39 → AOLOB39 → AOLOBt	311
ESALGs → ESALG9 → TGLFW9 → BJCOO9 → BJCOOt	173

Mediterranean paths

Table 22: Mediterranean paths

Path	Total flow in FFE
ESALGs → ESALG113 → MACAS113 → MACASt	180
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → ESALG75 → ESALG113 → MACAS113 → MACASt	148
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALYt	120
EGALYs → EGALY116 → LBBEY116 → LBBEYt	45

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	41
ITGITs → ITGIT12 → ITSAL12 → ITGOA12 → ITGOAt	36
ESALGs → ESALG75 → DZORN75 → ITSAL75 → TNSFA75 → TNSFAt	12
ESALGs → ESALG2 → DZORN2 → DZORNt	35
EGPSDs → EGPSD116 → EGALY116 → EGALYt	29
TRMERs → TRMER41 → SYLTK41 → EGPSD41 → EGPSDt	28
PTLEIs → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSDt	28
ITSALs → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → ESALG75 → ESALGt	22
ITSALs → ITSAL101 → EGPSD101 → EGPSDt	20
TNTUNs → TNTUN53 → DZALG53 → DZALGt	19
PTLEIs → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL12 → ITGOA12 → ITGIT12 → ITGITt	18
TNTUNs → TNTUN125 → ITSAL125 → ITGOA125 → ITGOAt	16
EGALYs → EGALY8 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → MAPTMt	13
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → ESVGO75 → ESVGOt	11
UAODSs → UAODS88 → EGDAM88 → EGALY88 → EGALYt	10
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → DZORNt	9
SYLTKs → SYLTK41 → EGPSD41 → EGPSDt	7

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	7
TNSFAs → TNSFA75 → ESVGO75 → PTLEI75 → ESALG75 → ESALGt	7
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → ESALGt	6
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSDt	5
PTLEIs → PTLEI75 → ESALG75 → ESALGt	5
ITGITs → ITGIT12 → ITSAL12 → ITSAL75 → TNSFA75 → ESVGO75 → ESVGOt	4
ESALGs → ESALG75 → DZORN75 → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	4
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSALt	4
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA125 → TNTUN125 → TNTUNt	4
ITGITs → ITGIT12 → ITSAL12 → ITSAL75 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	4
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA75 → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSALt	3
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → DZORNt] 2.0; Any[EGPSDs → EGPSD101 → EGDAM101 → EGDAMt	1
ESALGs → ESALG75 → DZORN75 → ITSAL75 → ITSALt	1
TNTUNs → TNTUN125 → ITSAL125 → ITGOA125 → TNSFA125 → TNSFA46 → MAPTM46 → MAPTMt	1
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFAt	1

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
DZORNs → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSDt	1
EGPSDs → EGPSD41 → CYLMS41 → TRMER41 → SYLTK41 → SYLTKt	71
EGPSDs → EGPSD41 → CYLMS41 → TRMER41 → TRMERt	57
EGPSDs → EGPSD116 → EGALY116 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	26
EGPSDs → EGPSD116 → EGALY116 → EGALY88 → GRPIR88 → BGVAR88 → UAODS88 → UAODSt	22
LBBEYs → LBBEY116 → ILHFA116 → EGDAM116 → EGPSD116 → EGALY116 → EGALY88 → GRPIR88 → BGVAR88 → UAODS88 → UAODSt	16
MACASs → MACAS113 → MAPTM113 → ESALG113 → ESALGt	15
ITSALs → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	6
EGPSDs → EGPSD41 → CYLMS41 → CYLMSt	3
MACASs → MACAS113 → MAPTM113 → MAPTMt	1
EGALYs → EGALY116 → LBBEY116 → ILHFA116 → EGDAM116 → EGPSD116 → EGPSDt	37
EGPSDs → EGPSD64 → ILASH64 → ILASHt	10
TNTUNs → TNTUN125 → ITSAL125 → ITSAL101 → EGPSD101 → EGPSDt	8
DZALGs → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUN125 → ITSAL125 → ITSAL101 → EGPSD101 → EGPSDt	2
EGALYs → EGALY8 → EGDAM8 → TNSFA8 → TNSFA125 → TNTUN125 → TNTUNt	2

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
TNSFAs → TNSFA125 → TNTUN125 → ITSAL125 → ITSAL101 → EGPSD101 → EGPSDt	1
ITGOAs → ITGOA125 → TNSFA125 → TNSFA46 → MAPTM46 → MAPTMt	146
ESALGs → ESALG2 → DZORN2 → DZALG2 → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUNt	3
ESALGs → ESALG2 → DZORN2 → DZALG2 → DZALGt	33
ITGOAs → ITGOA125 → TNSFA125 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	8
DZALGs → DZALG2 → ESAGP2 → ESAGPt	7
SYLTKs → SYLTK41 → EGPSD41 → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL125 → ITGOA125 → ITGOAt	7
ITGOAs → ITGOA12 → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGPSDt	125
ESALGs → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	55
ESALGs → ESALG46 → TNSFA46 → TNSFA125 → TNTUN125 → TNTUNt	46
ESALGs → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY116 → LBBEY116 → LBBEYt	37
ESALGs → ESALG46 → TNSFA46 → TNSFAt	23
ESALGs → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → CYLMSt	4
ESAGPs → ESAGP2 → ESALG2 → ESALG46 → TNSFA46 → TNSFAt	3
ESAGPs → ESAGP2 → ESALG2 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALYt	2

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
ESAGPs → ESAGP2 → ESALG2 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGDAM8 → EGDAMt	2
ESAGPs → ESAGP2 → ESALG2 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → CYLMSt	1
TRMERs → TRMER41 → SYLTK41 → EGPSD41 → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL125 → ITGOA125 → ITGOAt	33
TRMERs → TRMER41 → SYLTK41 → EGPSD41 → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL12 → ITGOA12 → ITGIT12 → ITGITt	12
CYLMSs → CYLMS41 → TRMER41 → SYLTK41 → EGPSD41 → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL12 → ITGOA12 → ITGIT12 → ITGITt	8
ITSALs → ITSAL125 → ITGOA125 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	59
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL125 → ITGOA125 → TNSFA125 → TNTUN125 → TNTUNt	46
TNTUNs → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	30
EGALYs → EGALY8 → EGDAM8 → TNSFA8 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	25
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL125 → ITGOA125 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALGt	22
MAPTMs → MAPTM113 → ESALG113 → MACAS113 → MACASt	22

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
EGALYs → EGALY8 → EGDAM8 → TNSFA8 → TNSFA75 → ESVGO75 → PTLEI75 → PTLEIt	9
TNSFAs → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	9
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL125 → ITGOA125 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZBJA53 → DZBJAt	4
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → EGALYt	15
ITGITs → ITGIT12 → ITSAL12 → ITSAL75 → TNSFA75 → TNSFA8 → CYLMS8 → CYLMS41 → TRMER41 → TRMERt	4
ITGITs → ITGIT12 → ITSAL12 → ITSAL75 → TNSFA75 → TNSFAt	1
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAMt	1
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL12 → ITGOA12 → ITGOAt	48
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → UAODS88 → UAODSt	30
EGPSDs → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → GRPIRt	15
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → GRPIRt	12
ITGOAs → ITGOA12 → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → UAODS88 → UAODSt	11
EGDAMs → EGDAM64 → EGPSD64 → EGPSDt	8

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	8
EGPSDs → EGPSD101 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → TNSFAt	4
CYLMSs → CYLMS8 → EGALY8 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → ESALG46 → ESALGt	1
LBBEYs → LBBEY116 → ILHFA116 → EGDAM116 → EGPSD116 → EGPSDt	99
ESALGs → ESALG2 → DZORN2 → DZALG2 → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUN125 → ITSAL125 → ITSAL101 → EGPSD101 → EGPSD64 → ILASH64 → ILASHt	56
ILHFAs → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → MAPTMt	42
ITGOAs → ITGOA125 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALGt	20
MACASs → MACAS113 → MAPTM113 → ESALG113 → ESALG2 → DZORN2 → DZALG2 → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUN125 → ITSAL125 → ITSAL101 → EGPSD101 → EGPSDt	14
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGPSD64 → ILASH64 → ILASHt	14
MACASs → MACAS113 → MAPTM113 → ESALG113 → ESALG2 → DZORN2 → DZALG2 → DZALGt	13
ESAGPs → ESAGP2 → ESALG2 → DZORN2 → DZALG2 → DZALGt	7
ILHFAs → ILHFA116 → EGDAM116 → EGPSD116 → EGPSDt	4

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
LBBEYs → LBBEY116 → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → MAPTMt	2
ILHFAs → ILHFA116 → EGDAM116 → EGDAM101 → ITSAL101 → ITSAL75 → TNSFA75 → ESVGO75 → ESVGOt	1
ILASHs → ILASH64 → EGDAM64 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	51
ILASHs → ILASH64 → EGDAM64 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → MAPTMt	25
ILASHs → ILASH64 → EGDAM64 → EGPSD64 → EGPSDt	23
ILASHs → ILASH64 → EGDAM64 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → ESALG46 → ESALGt	18
ILHFAs → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → ESALG46 → ESALGt	17
ILASHs → ILASH64 → EGDAM64 → EGDAM101 → ITSAL101 → ITSAL12 → ITGOA12 → ITGIT12 → ITGITt	16
LBBEYs → LBBEY116 → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → ESALG46 → ESALGt	2
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY88 → GRPIR88 → BGVAR88 → UAODS88 → UAODSt	23
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → DZORN2 → DZALG2 → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUNt	17
TNTUNs → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESALG2 → ESALGt	2
EGPSDs → EGPSD116 → EGALY116 → LBBEY116 → LBBEYt	86

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
PTLEIs → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → EGALYt	36
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	16
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → LBBEY116 → LBBEYt	16
PTLEIs → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	8
ESVGOs → ESVGO75 → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → EGALYt	6
PTLEIs → PTLEI75 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	3
EGPSDs → EGPSD116 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	3
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY116 → LBBEY116 → LBBEYt	44
BGVARs → BGVAR88 → UAODS88 → EGDAM88 → EGDAM64 → EGPSD64 → EGPSDt	40
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	26
BGVARs → BGVAR88 → UAODS88 → EGDAM88 → EGALY88 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	21

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
UAODSs → UAODS88 → EGDAM88 → EGALY88 → EGALY116 → LBBEY116 → ILHFA116 → ILHFAt	6
ITGITs → ITGIT12 → ITSAL12 → ITSAL101 → EGPSD101 → EGPSD41 → CYLMS41 → CYLMSt	4
ESALGs → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → EGALYt	190
ESALGs → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGDAM101 → EGDAM88 → EGALY88 → GRPIR88 → GRPIRt	76
MACASs → MACAS113 → MAPTM113 → ESALG113 → ESALG75 → DZORN75 → ITSAL75 → ITSAL101 → EGPSD101 → EGPSD116 → EGALY116 → EGALYt	16
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY88 → GRPIR88 → BGVAR88 → BGVARt	9
LBBEYs → LBBEY116 → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	28
ILHFAs → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	5
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → EGALY8 → EGALY88 → GRPIR88 → GRPIRt	7
PTLEIs → PTLEI75 → ESALG75 → ESALG2 → DZORN2 → DZALG2 → DZALGt	50
PTLEIs → PTLEI75 → ESALG75 → ESALG2 → DZORN2 → DZALG2 → DZALG53 → DZBJA53 → DZSKI53 → TNTUN53 → TNTUNt	34

Continued on next page

Table 22: Mediterranean paths (Continued)

Path	Total flow in FFE
TNTUNs → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESALG2 → ESALG113 → MACAS113 → MACASt	12
MAPTMs → MAPTM46 → ESALG46 → TNSFA46 → TNSFA8 → CYLMS8 → CYLMS41 → TRMER41 → TRMERt	41
TRMERs → TRMER41 → SYLTK41 → EGPSD41 → EGPSD101 → EGDAM101 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → MAPTMt	11
ILHFAs → ILHFA116 → EGDAM116 → EGDAM8 → TNSFA8 → TNSFA46 → MAPTM46 → ESALG46 → ESALG113 → MACAS113 → MACASt	84
MACASs → MACAS113 → MAPTM113 → ESALG113 → ESALG2 → DZORN2 → DZORNt	13
MAPTMs → MAPTM113 → ESALG113 → ESALG2 → DZORN2 → DZALG2 → DZALGt	3
UAODSs → UAODS88 → EGDAM88 → EGDAM116 → EGPSD116 → EGPSDt	105
ILASHs → ILASH64 → EGDAM64 → EGDAM8 → TNSFA8 → TNSFA125 → TNTUN125 → TNTUN53 → DZALG53 → DZALG2 → ESAGP2 → ESAGPt	9