NTNU
Innovation and Creativity

Optimization of water-borne crude oil transport

Karsten Dånmark Vatn

Master of Science in Engineering Cybernetics
Submission date: June 2007
Supervisor: Bjarne Anton Foss, ITK
Co-supervisor: Kjetil Fagerholt, NTNU-IØT

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

The purpose of this work is to study a real, complex problem of world-wide waterborne transport of crude oil and to develop, implement and test a heuristic solution method for optimising routing decisions for the crude vessels involved.

Main contents:
1. Presentation of the studied planning problem.
2. A presentation of the proposed heuristic for solving the problem.
3. Implementation of the proposed heuristic.
4. Presentation of test results of the heuristic.
5. Discussion of the heuristic and the test results with regard to applicability for solving real-life problems.

Assignment given: 18. January 2007
Supervisor: Bjarne Anton Foss, ITK

Optimization of water-borne crude oil transport

Report

Karsten Dånmark Vatn [1]

Department of Engineering Cybernetics
Norwegian University of Science and Technology—NTNU

June 11, 2007

[1]Email: karstend@stud.ntnu.no

Preface

This report is the result of work on a ship scheduling and routing problem. The work was carried out at the Norwegian University of Science and Technology under the Department of Engineering Cybernetics.

Much of the time used working on this assignment has been spent gaining knowledge of TurboRouter and developing its existing software. With limited experience with regards to both operational research and object oriented programming, a great deal of time and many late nights have been spent acquiring the necessary knowledge.

I want to give a special thanks to PhD scholar Jarl Eirik Korsvik for his priceless help with TurboRouter.

I also want to thank my supervisors, Professor Bjarne A. Foss, Kjetil Fagerholt, for great support and motivation.

The source code in TurboRouter is confidential, and belongs to MARINTEK. If examining this code is found necessary please contact Jarl Eirik Korsvik.

Trondheim, June 11, 2007

Karsten D. Vatn

Abstract

A ship scheduling problem in optimization of water-borne crude oil transportation has been investigated. The classic optimization problem the most closely related to the problem at hand is the Multi-Vehicle-Pick-up-and-Delivery Problem with Time Windows (m-PDPTW). In addition to the basic characteristics of the m-PDPTW, the studied problem has an additional degree of freedom due to having pick-ups and deliveries that are not matched. This extra freedom gives new possibilities when creating effective heuristics when dealing with transportation problems. The studied problem has been presented in relation to carefully selected background literature. On this basis a proposed heuristic has been developed, and implemented using some already existing structures in the commercial decision support system TurboRouter.

The studied problem is an industrial shipping problem, an operational mode where the shipper owns the cargo to be transported. No income is therefore made directly from transporting goods. Therefore the objective function chosen was net income, which in this mode is the same as minimizing the net expenses.

A multi-start local search with pre-matching of pick-ups and deliveries heuristic was chosen based on an assessment of problem size, problem type, real life applicability and existing software. This heuristic consists of three main parts. First the pick-ups and deliveries are matched and merged in a pre-matching heuristic, and then a large number of initial solutions are generated by an insertion heuristic. The best initial solutions are then improved by a local search. Two strategies were developed for pre-matching and then tested. The one with the best test results was subsequently used in the heuristic.

This multi-start local search with pre-matching of pick-ups and deliveries heuristic has been subject to rigorous testing and was compared to a single-start local search and multiple initial solutions heuristic. The solutions generated by the multi-start local search heuristic were superior compared to those of the other heuristics, but the computation time necessary was high and higher than those of the heuristics which it was compared to. This high computation time is partially believed to be a result of flexible data sets resulting in broad solution spaces. In addition some computationally expensive heuristics were deployed, increasing the computation time. In real life applications, finding a solution relatively quickly is of importance. Therefore the heuristic may need to be simplified and used on "tighter" data sets than some sets used in testing to be real life applicable.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

$\mathcal{A}_A$ All arcs in the model.
$\mathcal{A}_A = \{(imjnk) : i \in \mathcal{N}_A, m \in \mathcal{T}_{Wi}, j \in \mathcal{N}_A, n \in \mathcal{T}_{Wj}, k \in \mathcal{K}_{imjn}, (imjnk)exists\}$

$\mathcal{A}_{Av}$ All arcs vessel $v \in \mathcal{V}_A$ can physically sail.
$\mathcal{A}_v = \{(imjnk) : i \in \mathcal{N}_{Av}, m \in \mathcal{T}_{Wi}, j \in \mathcal{N}_{Av}, n \in \mathcal{T}_{Wj}, k \in \mathcal{K}_{imjn}, (imjnk)exists\}$

$\mathcal{C}_A$ The set of all crude grades

$\mathcal{K}_{imjn}$ The set of routing options between node $i \in \mathcal{N}_A$ in time window $m \in \mathcal{T}_{Wi}$ and node $j \in \mathcal{N}_A$ in time window $n \in \mathcal{T}_{Wj}$ if $(imjn)$ exists

$\mathcal{N}_A$ The set of all nodes

$\mathcal{N}_{Av}$ The set of all nodes $\mathcal{N}_A$ available for vessel v

$\mathcal{N}_D$ The set of discharge ports (delivery nodes)

$\mathcal{N}_{Dv}$ The set of discharge ports $\mathcal{N}_D$ available for vessel v

$\mathcal{N}_P$ The set of loading ports (pick-up nodes)

$\mathcal{N}_{Pv}$ The set of loading ports $\mathcal{N}_P$ available for vessel v

$\mathcal{T}_{Wi}$ The set of time windows for port $i \in \mathcal{N}_A$

$\mathcal{V}_A$ The set of all vessels

$\mathcal{V}_{Ni}$ The set of all vessels $v \in \mathcal{V}_A$ that generally can enter port $i \in \mathcal{N}_A$

$\mathcal{V}_S$ The set of spot charged vessels

$\mathcal{V}_T$ The set of company owned vessels

c Crude grade

C_{FUEL} Fuel price in kUSD/ton

$C_{LEGijkv}$ Cost for sailing between port $i \in \mathcal{N}_A$ and port $j \in \mathcal{N}_A$ via route k for vessel $v \in \mathcal{V}_T$ in kUSD. This price includes pure sailing in ballast, or loaded and costs for berthing and deberthing

C_{PORTiv} Port fee in port $i \in \mathcal{N}_A$ for vessel $v \in \mathcal{V}_T$

$d(v)$ Destination node for vessel v

D_{ijk} Distance between port $i \in \mathcal{N}_A$ and port $j \in \mathcal{N}_A$ on route option $k \in \mathcal{K}_{imjn}$

F_{IDLEv} Fuel consumption for vessel $v \in \mathcal{V}_T$ when waiting in tons/day

F_{PORTv} Fuel consumption in port for vessel $v \in \mathcal{V}_T$ in tons/day

i, j Index of the ports

l_{imcv} Load of crude grade $c \in \mathcal{C}_A$ on board vessel $v \in \mathcal{V}_A$ as it departs from port $i \in \mathcal{N}_{Av}$ in time window $m \in \mathcal{T}_{Wv}$

m, n Time windows

$o(v)$ Origin node for vessel v

Q_{cim} Demand in discharge port $i \in \mathcal{N}_D$, time window $m \in \mathcal{T}_{Wi}$ and crude grade $c \in \mathcal{C}_A$. For pick-up ports ($i \in \mathcal{N}_P$), Q_{cim} is the amount of a certain crude grade to be picked up in the certain time window

T_{Bij} Berthing time in between port i and j. This includes deberthing time in port i and berthing time in port j

t_{imv} In nodes $i \in \mathcal{N}_{Pv} \cup \mathcal{N}_{Dv}$: The time for start of loading/discharge for vessel v in time window $m \in \mathcal{T}_{Wi}$. If $i = o(v)$ or $i = d(v)$: The time for start and end of service during the planning horizon

T_{MNim} The opening time of time window $m \in \mathcal{T}_{Wi}$ in port $i \in \mathcal{N}_A$

T_{MXim} The closing time of time window $m \in \mathcal{T}_{Wi}$ in port $i \in \mathcal{N}_A$

T_{Qicv} Load/Discharge rate (days/ktons) in port $i \in \mathcal{N}_{Pv} \cup \mathcal{N}_{Dv}$ for crude type $c \in \mathcal{C}_A$ and vessel $v \in \mathcal{V}_A$

T_{Sijkv} Sailing time on leg $(imjnkv) \in \mathcal{A}_{Av}$ for vessel $v \in \mathcal{V}_A$. Berthing time T_{Bij} is included

v Vessel

V_{CAPVv} The volume capacity of vessel v

V_{CAPWv} The weight capacity of vessel v

x_{imjnkv} Boolean variable. $x_{imjnkv} = 1$ if vessel v sails from port $i \in \mathcal{N}_v$ in time window $m \in \mathcal{T}_{Wi}$ to port $j \in \mathcal{N}_v$ in time window $n \in \mathcal{T}_{Wj}$ via route option $k \in \mathcal{K}_{imjn}$. $x_{imjnkv} = 0$ otherwise.

Acronyms

TSP Travelling Salesman Problem

VRP Vehicle Routing Problem

VRPTW Vehicle Routing Problem with Time Window

PDP Pick-up and Delivery Problem

PDPTW Pick-up-and-Delivery Problem with Time Window

m-PDPTW Multi-Vehicle-Pick-up-and-Delivery Problem with Time Window

I1 Solomon's insertion heuristic

k-opt k-opt exchange heuristic

2-opt* 2-opt* exchange heuristic

Or-opt Or-opt exchange heuristic

RTS Reactive Tabu Search

NPC NP-Complete

Chapter 1

Introduction

Continuous growth in the world population, rising standard of living, road congestion and increasing globalization, are all major factors increasing the already massive use of seaborne shipping which is the major mode of transportation caused by international trade today. With the monopoly the shipping industry has on long distance transportation of large volumes, mainly between continents, it is expected to se a further increase in the total international seaborne trade which has increased by 87% since 1987 according to Christensen et al. (2007).

By the end of 2003 the cargo carrying capacity of the world exceeded 857 million tons, an increase of more than 25% from 1980. This increase in capacity has been followed by an increase in utilization as well. The utilization of the worlds fleet has increased from 5.4 tons carried per deadweight ton in 1980 to 7.2 tons in 2003, see Christensen et al. (2007). A vessel used in long distance transportation may easily cost tens of millions of dollars to buy, and tens of thousands of dollars in daily operating costs. It goes without saying that you want to use these vessels as efficiently as possible and only slight improvements may yield great benefits.

Consolidations in the manufacturing sector and tough competition between shipping companies during the last decades have given increased market power to the cargo owners. This imbalance has resulted in mergers of many shipping companies. As a result of this fleets have become larger, and more information is spread across larger and geographically separated areas. This increased complexity in addition to increased competition between shipping companies has rendered manual planning methods used in the past insufficient. These are some main reasons for the need for decision support systems. Further arguments are given in Christensen et al. (2007).

Maritime transportation problems can be divided into three levels with regard to their planning horizon according to (Christensen et al. 2007, page 8-9). These are strategic,

tactical and operational problems. Short term scheduling is the topic of this report. Therefore strategic decisions such as optimal fleet and marine supply chains, described in Christensen et al. (2004), are not discussed. Neither are operational problems such as cruising speed selection, ship loading, and environmental routing.

There are three general operation modes in shipping, *industrial*, *tramp* and *liner*, see Ronen (1993). In industrial shipping the cargo owner controls the ships, minimizing total costs is therefore the something the industrial shippers strive towards. Tramp ships follow the cargo like taxi cabs, normally having some contract cargoes to carry in addition to trying to maximize the profit from other optional cargo. Liners are similar to buses, in the way that they operate according to a published itinerary and schedule. In this report we consider an industrial or term shipping problem, meaning that the operator owns the cargo. Minimizing the cost is therefore the main issue. The fleets of industrial operators often have the same characteristics as mentions above with regards to fleet size, and widespread information. In addition they compete with other cargo owners delivering the same product. Industrial operations are explained in greater detail in Christensen et al. (2007).

Using the definitions in (Ronen 1993, page 325), *routing* is defined as the assignment of sequences of ports to be visited by the vessels, and *scheduling* is defined as assigning times (or time windows) to the events on the vessels route. A voyage is described in (Christensen et al. 2007, page 13) as a sequence of port calls, starting with the port where the vessel loads its first cargo and ending where the vessel unloads its last cargo and becomes available again. It is also stated that a voyage may include multiple loading ports and multiple unloading ports.

The problem at hand is to the authors knowledge part of a new area of research within maritime transportation, a field with observed significant growth in research according to (Christensen et al. 2007, 194). Finding literature on similar problems like the one at hand, has not been possible, but it resembles the Multi-Vehicle-Pick-up-and-Delivery Problem with Time Window (m-PDPTW) for which numerous heuristics have been developed according to Brønmo et al. (2007). Only in the m-PDPTW orders consist of a load to be picked up at one location and delivered at another, within time constraints called time windows. In the problem at hand, a load is available to be picked up at one location, and a load is to be delivered at one location within time windows. But which pick-up going where is not decided. This adds another dimension with regards to the traditional m-PDPTW.

The problem at hand is a real world problem. It consists of creating the optimal (minimum cost) sailing plan for a set of vessels servicing a set of pick-ups and deliveries that are not matched in the way explains above. Ship scheduling problems are complex and tightly constrained. Canals can only take limited sized ships, a cargo normally needs to

be picked up within a time window, not all ships may enter any port also due to size restrictions, and the ships are of different size and have different sailing speed this only being some factors to be considered in optimization. This complete problem description has been simplified to some extent to fit the scope of this report.

In theory, one would be interested in finding the optimal sailing plan with regards to cost. Due to constraints on time and computation power, finding the best allocation of cargoes on a fleet of vessels may not be possible. Therefore heuristic are employed. Heuristics do not usually find the best allocation of cargoes, but the solution provided by the right heuristic for a specific problem will normally give a good solution relatively quickly.

The goal of the report is to develop, implement and test a heuristic for the problem at hand. A pre-matching of pick-ups and deliveries followed multi-start local search heuristic is presented. Two slightly different types of pre-matching followed by multi-start local search heuristic are tested, to find the best one of the two. The best pre-matching heuristic is then tested thoroughly to investigate its characteristics and real world applicability.

The rest of the report is organized as follows. In Chapter 2 background literature and theory on similar problems and solution methods, including complete methods and heuristics, are presented. The traits and most importantly the strengths and weaknesses of the different methods are then discussed. Chapter 3, describes first the outline of the complete problem at hand, then the simplifications made with regards to the scope of the report are stated. The problem is then discussed considering the background literature, and on this basis the solution method is decided. The framework of multi-start local search heuristic is presented in Chapter 4, including the different local search operators used. Chapter 5 describes the characteristics of the data set, the overall description of the solution, and the algorithms for matching deliveries and pick-ups.

Details on the testing of the heuristic solutions are described along with the results in Chapter 6. Next, in Chapter 7 the solution methods used, the choices made, and their real world applicability are discussed. The conclusions follow in Chapter 8. Several ideas and possible areas of further work and development are included in Chapter 9. Due to the complexity of the problem, and sheer workload necessary to construct software capable of solving the problem at hand without pre-matching of pick-ups and deliveries. These ideas have not been put to life in this report. The ideas are discussed, at the end of the chapter.

Finally some clarifying remarks with regards to terms used throughout this report: In the literature review the term vehicle is used, while the terms vessel or ship is used when describing the problem at hand. This because most of the literature on Vehicle Routing Problem (VRP)s deals with land based transportation, while the problem at hand is a problem of water-borne transportation. The terms vessel and ship are used interchange-

ably throughout the report, even though a vessel is a broader term than ship. Cargo and order are also used interchangeably, in this report they are defined as a certain amount of goods (crude oil) to be transported from one port to another. A pick-up is a certain amount of crude oil to be loaded in a specific port, without having a designated unloading port. Deliveries are certain amounts of crude oil to be unloaded in specific ports.

Chapter 2

Literature review

This chapter is a brief introduction to classical routing and scheduling problems and heuristic solution methods used to solve them. The literature has been chosen on basis of the outline of the problem at hand, which has many similarities with the m-PDPTW. On this basis, a set of classical routing problems have been reviewed, after first giving a brief outline of NP-Complete (NPC) problems. In order to best determine the way of solving the problem, some common solution methods are then outlined. Both complete methods and heuristics have been examined.

These methods and the ideas behind them will serve as introduction to, and motivation behind the solution strategy. After the problem at hand has been thoroughly described in Chapter 3, the traits of the problem and possible solution methods are discussed, followed by an outline of which strategy to use in solving the problem. This strategy will be further developed and refined in Chapter 4 and Chapter 5.

2.1 NP-Complete problems

NPC problems is a problem class, believed to be intractable by most computer scientists since a give range of NPC problems have been studied to date, without there being discovered a polynomial time solution to any of them, as stated in (Cormen et al. 2001, page 968). NPC problems also have the trait that if one was to solve one such problem in polynomial time, all NPC problems would have a polynomial time solution.

Problems are often divided into three classes. P, NP and NPC. The class P consists of problems that are solvable in polynomial time, specifically meaning that they can be solved in time $O(n^k)$, for some constant k, where n is the size of the problem. The class NP consists of problems that are "verifiable" in polynomial time. NP-completeness is almost always proved by reduction. This is done by reducing the problem at hand to a

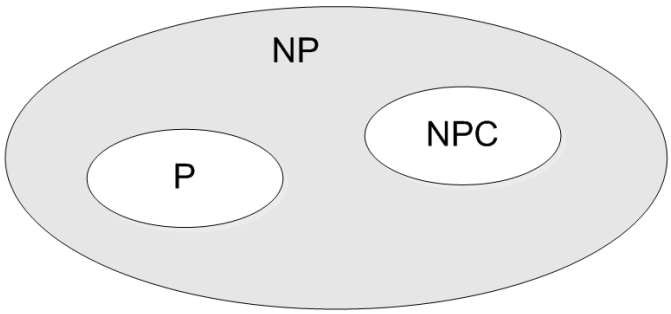

Figure 2.1: Relationship among complexity classes

problem already proved to be NPC.

The relationship between the complexity classes is by most researchers believed to be as shown in Figure 2.1. If however a NPC problem was to be solved in polynomial time P = NP. For more information on the complexity classes, see Cormen et al. (2001).

2.2 Classical routing problems

When seeking a solution to an optimization problem, it is crucial to be able to describe the problem at hand with regards to real-world feasibility of any solution derived. There is a broad base of literature concerning optimization problems, of different types and with different constraints. Knowing the nature of the problem at hand will help when searching for relevant literature.

In this chapter, a carefully selected range of routing and scheduling problems important for the work behind this report will be described. These problems have been chosen because they are closely related to the optimization problem at hand, which is explained in great detail in Chapter 3. The general optimization problem that the most closely resembles the problem at hand is the m-PDPTW.

Different types of problems pose different difficulties when trying to solve them. The problems described here are described in a hierarchic order, by starting with the Travelling Salesman Problem (TSP), and then generalizing the TSP to the VRP, which in turn is generalized until finally ending up at the Pick-up-and-Delivery Problem with Time Window (PDPTW).

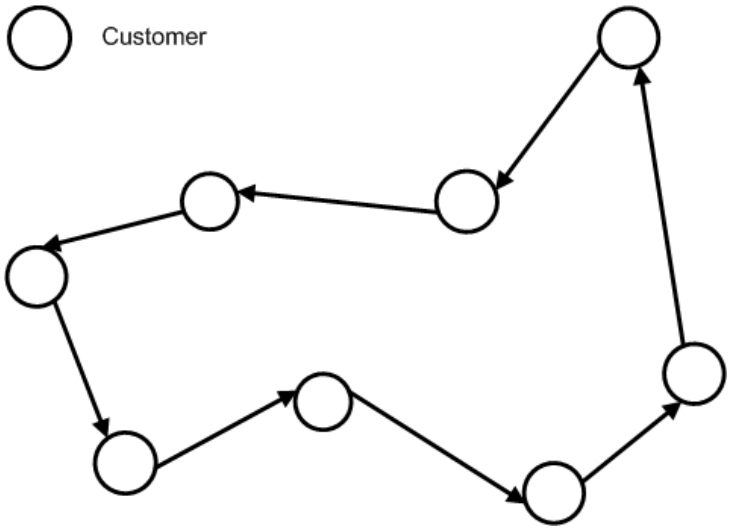

Figure 2.2: Example of a Travelling Salesman Problem (TSP)

2.2.1 The Travelling Salesman Problem (TSP)

Informal description of the TSP

This problem is quite easily described; given a set of nodes, the goal is to build a closed tour that goes through each of the nodes, once and only once, and returning to the starting node when finished (the depot). This while at the same time minimizing the distance travelled (or more generally a cost depending on time spent and distance travelled).

The nodes may represent customers to be visited, therefore the name Travelling Salesman Problem. Minimizing the cost of the tour, the TSP may be reformulated as finding the minimum cost Hamiltonian cycle in a graph where the nodes represent customers, and the arcs are valued. A Hamiltonian cycle is defined in (Cormen et al. 2001, page 967) as a directed graph $G = (V, E)$ that contains each vertex in V. The cost usually represents the distance between the two nodes, connected by the arc.

The TSP may be extended to a problem with multiple salesmen. Now the problem consists of a set of customers to be visited by two or more salesmen. The customers must still only be visited once, and the salesmen must start and end their tours at the same node, this node representing the depot. The problem consists of determining how the salesmen should divide the customers between each other, making the total costs of the system minimal.

Mathematical model of the TSP

Let x_{ij} be a Boolean variable, where $x_{ij} = 1$ if node j is visited directly after node i and $x_{ij} = 0$ otherwise. The cost of going from node i to node j is denoted C_{ij}. The goal is to minimize the total cost:

$$z = min \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij} \tag{2.1}$$

Only one arc can leave each of the n nodes, this constraint is given by:

$$\sum_{j=1}^{n} x_{ij} = 1, \ i = 1, ..., n \tag{2.2}$$

Each node can also only entered by one arc:

$$\sum_{i=1}^{n} x_{ij} = 1, \ j = 1, ..., n \tag{2.3}$$

(2.2) and (2.3) together deal with the fact that the salesman only can visit each customer once, and only once. Together with (2.1), they make up an assignment problem that allows sub-tours. The TSP does not allow sub-tours, therefore additional constraints are included. For a non empty set $\mathcal{S}$ of nodes, there are strictly less than $|\mathcal{S}|$ arcs linking these nodes.

$$\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1 \tag{2.4}$$

In addition to (2.4), there has to exist at least one arc linking the nodes in set $\mathcal{S}$ to the nodes in $\overline{\mathcal{S}}$:

$$\sum_{i \in \mathcal{S}} \sum_{j \in \overline{\mathcal{S}}} x_{ij} \geq 1 \tag{2.5}$$

2.2.2 The Vehicle Routing Problem (VRP)

Informal description of the VRP

The VRP can according to (Bräysy et al. 2003*a*, page 3) be described as the problem of designing least cost routes from one depot to a set of geographically scattered points, or

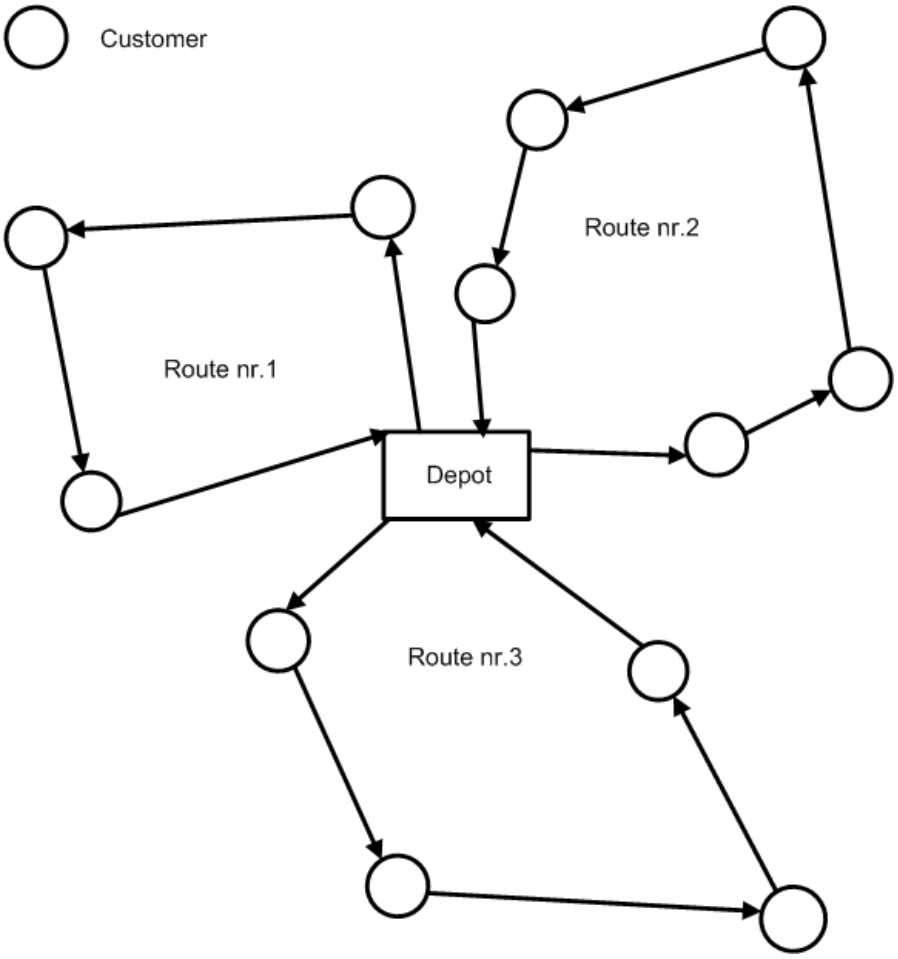

Figure 2.3: Example of a Vehicle Routing Problem (VRP)

nodes. These nodes may represent cities, stores, warehouses, customers etc. What the nodes represent depends on the problem at hand. The routes must be designed in a way that each node is visited only once by one vehicle. Routes must start and end at the depot, and the capacity of each vehicle must not be exceeded by the sum of the demands along one route. Different real-world problems have different constraints, a vessel cannot show up at any port at any given time, two different load may be mixable, two other may not etc. The VRP model can be extended by side constraints, to create real-world applicable methods and concepts for VRPs of different types.

Mathematical model of the VRP

The goal is to minimize the total cost, seen in (2.6). x_{ijv} is a Boolean variable, where $x_{ijv} = 1$ if node j is visited directly after node i by vehicle v and $x_{ijv} = 0$ otherwise. The cost of going from node i to node j is denoted C_{ij}. $\mathcal{V}$ is the set of vehicles, $\mathcal{N}_A$ is the set of nodes and $\mathcal{A}$ is the set of arcs. All the nodes except the depot, which is represented as node 0 and $n+1$, are included in the set of customers $\mathcal{N}_C$.

$$z = min \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}} C_{ij} x_{ijv} \tag{2.6}$$

Each arc must be used by one and only one vehicle:

$$\sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_A} x_{ijv} = 1, \ \forall i \in \mathcal{N}_C \tag{2.7}$$

Each customer (i) has a demand D_i. Stating that the vehicles are of the same type and have the same capacity Q, this capacity cannot be exceeded:

$$\sum_{i \in \mathcal{N}_C} D_i \sum_{j \in \mathcal{N}_A} x_{ijv} \leq Q, \ \forall v \in \mathcal{V} \tag{2.8}$$

Every vehicle must start their tour at the depot:

$$\sum_{j \in \mathcal{N}_A} x_{0jv} = 1, \ \forall v \in V \tag{2.9}$$

A vehicle can only leave node $h \in \mathcal{N}_C$ if and only if it enters node h:

$$\sum_{i \in \mathcal{N}_A} x_{ihv} - \sum_{j \in \mathcal{N}_A} x_{hjv} = 0, \ \forall h \in \mathcal{N}_C, \ \forall v \in \mathcal{V} \tag{2.10}$$

All the vehicles have to end at node $n+1$ which represents the depot.

$$\sum_{i \in \mathcal{N}_C} x_{i(n+1)v} = 1, \ \forall v \in \mathcal{V} \tag{2.11}$$

2.2.3 The Pick-up and Delivery Problem (PDP)

Informal description of the PDP

The Pick-up and Delivery Problem (PDP) is according to (Bräysy et al. 2003*b*, page 3) defined as a planning problem where one has one fleet of vehicles for serving a set of transportation requests with given capacity and start and end locations. Each request specifies the size of the load to be transported, the location where the load is to be picked

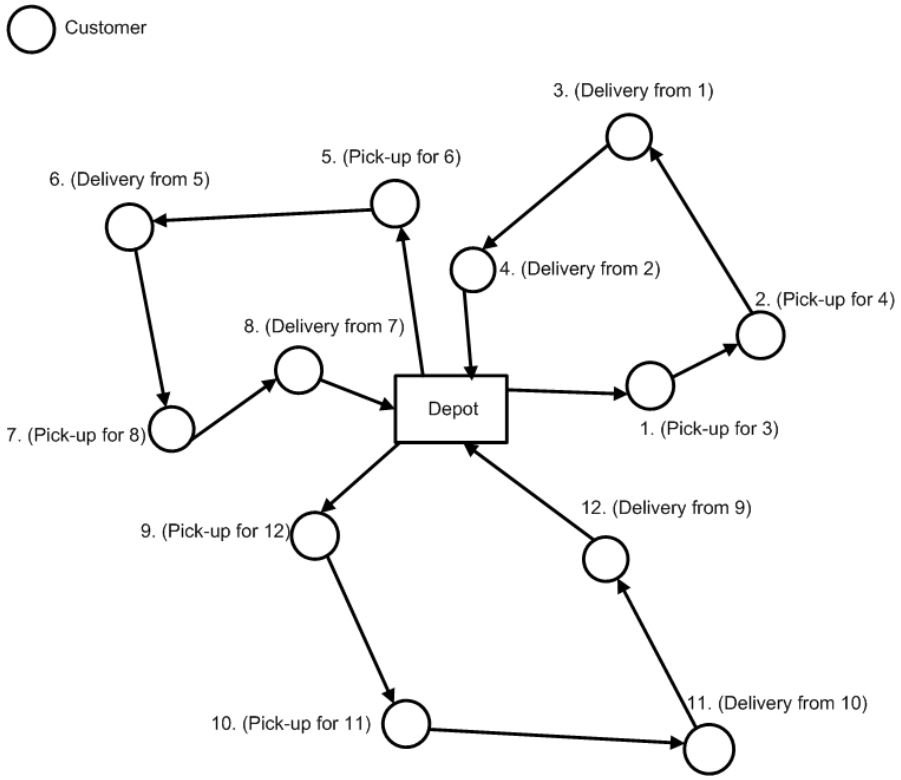

Figure 2.4: Example of a Pick-up and Delivery Problem (PDP)

up, and the location where it is to be delivered. Each load has to be transported by one vehicle from its origin to its destination, without any transhipment at other locations. Figure 2.4 shows an example of a PDP with only one depot present. The fleet may be stationed at multiple depots, but this possibility is not considered when deriving the mathematical model presented below. The goal is to minimize the total transportation costs. Pick-ups must be performed before their respective deliveries.

Mathematical model of the PDP

This model is represented in Savelsbergh & Sol (1995). It supposes that each customer i requests the transportation of goods from an origin i^+ to a destination i^-. $\mathcal{N}_\mathcal{C}$ is the set of customers, in other words $i \in \mathcal{N}_\mathcal{C}$. $\mathcal{N}^+$ is the set of customers that ask for a pick-up and $\mathcal{N}^-$ is the set of customers that ask for a delivery. $\{0\}$ is the depot. $\mathcal{N}_\mathcal{A} = \{0\} \cup \mathcal{N}^+ \cup \mathcal{N}^-$. $\mathcal{V}$ is the set of vehicles and $\mathcal{A}$ is the set of arcs. x_{ijv} is a Boolean variable, where $x_{ijv} = 1$ if vehicle $v \in \mathcal{V}$ uses arc $(i, j) \in \mathcal{A}$, and $x_{ijv} = 0$ otherwise.

Minimizing the following function is the goal:

$$z = min \sum_{(i,j)\in\mathcal{A}} \sum_{v\in\mathcal{V}} C_{ij} x_{ijv} \tag{2.12}$$

Each customer is to be visited by only one vehicle:

$$\sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} = 1,\ \forall i \in \mathcal{N}^+ \tag{2.13}$$

If a vehicle enters node i, it must leave it as well:

$$\sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} - \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} = 0,\ \forall i \in \mathcal{N}^+ \cup \mathcal{N}^-,\ v \in \mathcal{V} \tag{2.14}$$

The same vehicle must make the pick-up i^+ and the delivery i^-:

$$\sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} - \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ji^-v} = 0,\ \forall i \in \mathcal{N}_\mathcal{C},\ v \in \mathcal{V} \tag{2.15}$$

2.2.4 The Pick-up and Delivery Problem with Time Windows (PDPTW)

Informal description of the PDPTW

As described in (Ropke & Pisinger 2006, page 455), the PDPTW is a PDP with its requests and vehicles, with additional constraints on the timing of the pick-up's and deliveries. A request consists of picking up goods at one location and delivering these goods at another location. Two time windows are assigned to each request: a pick-up time window specifying when the goods can be picked up and a delivery time window that determines when the goods can be dropped off. Service times are also included in the PDPTW, this indicates the time that performing the pick-up or delivery will take. A vehicle may arrive at a pick-up or delivery before the start of a time window associated with that task specified at the certain location, but it has to wait until the start of the time window before initiating the operation. A vehicle may not arrive at a location after the end of the time window of the location.

Each request is assigned to a set of feasible vehicles, this enabling the modelling of situations where some vehicles cannot enter certain locations because of physical constraints related to the dimensions of the vehicle. The vehicles also have limited capacity and start and end its duties at given locations. The start and end locations need not be the same and for two vehicles these locations may be different.

The PDPTW is a generalization of the Vehicle Routing Problem with Time Window (VRPTW), which is NP-hard according to Ropke & Pisinger (2006). Therefore the PDPTW must also be NP-hard.

Mathematical model of the PDPTW

This model is the same as that presented in Chapter 2.2.3, with time windows as additional constraints. It supposes that each customer i requests the transportation of goods from an origin i^+ to a destination i^-. $\mathcal{N}_\mathcal{C}$ is the set of customers, in other words $i \in \mathcal{N}_\mathcal{C}$. $\mathcal{N}^+$ is the set of customers that ask for a pick-up and $\mathcal{N}^-$ is the set of customers that ask for a delivery. $\{0\}$ is the depot. $\mathcal{N}_\mathcal{A} = \{0\} \cup \mathcal{N}^+ \cup \mathcal{N}^-$. $\mathcal{V}$ is the set of vehicles and $\mathcal{A}$ is the set of arcs. x_{ijv} is a Boolean variable, where $x_{ijv} = 1$ if vehicle $v \in \mathcal{V}$ uses arc $(i,j) \in \mathcal{A}$, and $x_{ijv} = 0$ otherwise.

Minimizing the following function is the goal:

$$z = min \sum_{(i,j)\in\mathcal{A}} \sum_{v\in\mathcal{V}} C_{ij} x_{ijv}, \tag{2.16}$$

Each customer is to be visited by only one vehicle:

$$\sum_{v\in\mathcal{V}} \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} = 1, \ \forall i \in \mathcal{N}^+ \tag{2.17}$$

If a vehicle enters node i, it must leave it as well:

$$\sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} - \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ijv} = 0, \ \forall i \in \mathcal{N}^+ \cup \mathcal{N}^-, \ v \in \mathcal{V} \tag{2.18}$$

The same vehicle must make the pick-up i^+ and the delivery i^-:

$$\sum_{j\in\mathcal{N}_\mathcal{A}} x_{i^+jv} - \sum_{j\in\mathcal{N}_\mathcal{A}} x_{ji^-v} = 0, \ \forall i \in \mathcal{N}_\mathcal{C}, \ v \in \mathcal{V} \tag{2.19}$$

Given that d_i is the depart time from customer i, and $[T_{MNi^+}, T_{MXi^+}]$ $[T_{MNi^-}, T_{MXi^-}]$ are the time windows of customer i. The time windows can be added using the following equations:

$$T_{MNi} \leq d_i \leq T_{MXi}, \forall i \in \mathcal{N}^+ \cup \mathcal{N}^- \tag{2.20}$$

Given that t_{ij} is the travel time from node i to j:

$$d_{i^+} + t_{i^+i^-} \leq d_{i^-} \ \forall i \in \mathcal{N} \tag{2.21}$$

Given that t_{ij} is the travel time from node i to j:

$$x_{ijv} = 1 \Rightarrow d_i + t_{ij} \leq d_j \ \forall (i,j) \in \mathcal{A}, v \in \mathcal{V} \tag{2.22}$$

2.3 Complete solution methods

Complete methods are concerned with computing the optimal solution of a problem. If there are more than one equally optimal solution, complete methods will find one but not necessarily all solutions. As described earlier the PDPTW is NP-hard. According to (Cormen et al. 2001, page 986), these problems are at least as hard as NP problems which a verifiable in polynomial time. Significant progress in solving NP-hard optimization problems to optimality has been made in the recent decades. Researchers have for a long time struggled with this topic, and even with the advances that have been made, many problem types can only be solved for fairly small instances. In other words generating an optimal solution of a large problem of this type, may not be possible.

The objective of this report is to develop and use heuristics to solve the problem at hand. A selected few complete solution methods are included in the literature review to show that these types of methods exists and how they work, but also to point out the limitations of these methods.

2.3.1 Set partitioning

The m-PDPTW is mainly focused on which vehicle is to service which assignment. Every combination of assignments possibly assigned to each vehicle, is here called an alternative route. A set partition model can then be used to decide which alternative routes that should be chosen to service each assignment while minimizing the objective function.

$$z = min \sum_{i=1}^{n} c_i x_i \tag{2.23}$$

$$\sum_{i=1}^{n} a_{ij} x_{ij} = 1 \ \forall j \tag{2.24}$$

$$x \in 0, 1 \tag{2.25}$$

The set partition model is shown in (2.23), (2.24) and (2.25). To explain the different variables used, lets consider a m-PDPTW with n assignments (sets of pickups and deliveries). The cost function z to be minimized (2.23), consists of c_i which is the cost of alternative i. If the binary variable $x_i = 1$, this means that alternative i is used. If alternative i is not used $x_i = 0$. Every assignment must be fulfilled, therefore it is necessary to control this. (2.24) deals with this aspect as $a_{ij} = 1$ if alternative i consists of assignment j and $a_{ij} = 0$ otherwise.

Set partitioning models can be solved by commercial software like C-Plex and Xpress. If the problems are too complex though, constructing all the possible alternatives may not be possible. In some cases a column generation approach may be used.

2.3.2 Column generation

For the m-PDPTW, there exists an enormous amount of routing alternatives available even for moderately sized problems. This is the reason for its NP-hardness. In stead of creating all the alternatives, *Column generation* deals with this issue by generating these alternatives dynamically, when they are needed. Dumas et al. (1991) presents a column generation solution that is able to solve a m-PDPTW problem to optimality.

The idea behind column generation is dividing the problem into a master and a sub problem as described in Lübbecke & Desrosiers (2005). The master problem is composed by a sufficiently meaningful subset of variables. More variables are added only when needed. The master problem is solved by set partitioning which gives dual variables that in turn are used in the modified objective function in a sub-problem. The implicit search for a minimum reduced cost variable amounts to optimizing a sub problem. This being a shortest path problem solved with dynamic programming. The reduced cost variable is returned to the master problem, and it is used in its next iteration. If no reduced cost variable is found however, the linear relaxation of the master problem is finished. To make sure that the integer solutions of the original problem is found, a column generation within a branch-and-bound framework is embedded.

(Lübbecke & Desrosiers 2005, page 28) states that an off-the-shelf column generation software to solve large scale integer programs is within reach reasonably soon; the necessary building blocks are already available.

2.4 Heuristic solution methods

The drawbacks of the complete methods make it necessary to explore the search space in alternative ways. Heuristics may be employed to facilitate searches in extensive and complex datasets, where finding the global optimum is not feasible due to the complexity of the problem or constraints on the computation time.

For solving TSPs, VRPs, PDPs and PDPTWs there are two main types of heuristics. These are algorithms that build one or more initial solutions, often referred to as constructive heuristics, and algorithms that improve given solutions. Improving heuristics may be combined into heuristic frameworks like local search, and meta-heuristics. The

multi-start local search heuristic combines constructive and improving heuristics, performing a local search from several start points (initial solutions). The different problem types have different constraints and particularities that will affect the way the heuristics may be implemented, but the general ideas behind the heuristics can be viewed as general.

2.4.1 Constructive heuristics

Initial solutions are created by construction algorithms. These algorithms move through partial solutions, by choosing the value on one decision variable at the time. Construction algorithms usually stop when the first feasible solution is reached.

Savings heuristics

The savings heuristics developed by Clarke & Wright (1964) is one of the most well-known construction heuristics. Initially it starts with a solution where every customer is served directly by an individual route. By combining two of the routes, only servicing one customer each, there will be a cost reduction denoted by $S_{ij} = d_{i0} + d_{0j} - d_{ij}$. Here the distance from customer i to the depot is denoted as d_{i0}, and the distance from the depot to the customer j is denoted as d_{0j} and d_{ij} is the distance from customer i to customer j.

From Figure 2.5 the basic principle of the savings heuristic is demonstrated. The initial solution is shown on the left side. Here each of the customers i and j, are serviced by two separate routes. The cost reduction S_{ij} is calculated for every possible combination of two routes. If the savings heuristic finds that the total distance will be reduced and no constraints will be violated, the two routes that give the greatest reduction in cost are chosen. The new and improved route is created by removing arch $(i, 0)$ and $(0, j)$, the arch (i, j) is added. The saving heuristic procedure can be used iteratively. When combining routes, partial routes can be designed for all the customers at the same time, or one can add customers iteratively to a given route until the route if fully loaded.

Solomon's insertion heuristic (I1)

Solomon's insertion heuristic (I1) created by Solomon (1987) is also one of the most famous and used construction heuristics. Firstly a route with seeded customer is created. The seeding may be based on which customers are the farthest, or which customer has the earliest deadline (first closing time window). When the route has been initialized, two criteria $c_1(i, u, j)$ and $c_2(i, u, j)$ are used at every iteration to decide which new customer u to insert into the current partial route, between adjacent customers i and j on the route. The current route is denoted as $(i_0, i_1, i_2, ..., i_m)$, where i_0 and i_m represent

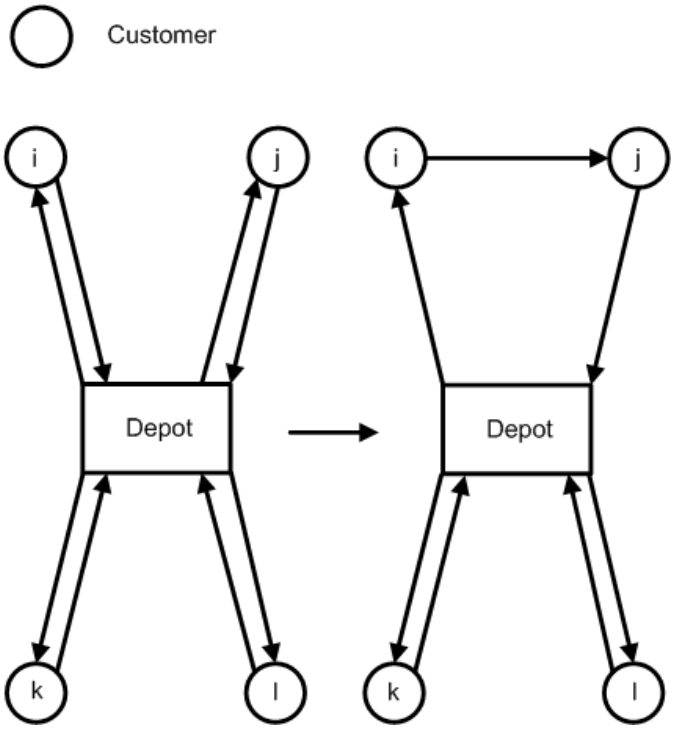

Figure 2.5: Savings heuristic

the depot[1].

For each non-routed customer, the best feasible insertion place in the emerging route is calculated as

$$c_1(i(u), u, j(u)) = min[c_1(i_{p-1}, u, i_p)],\ p = 1, ..., m$$

Next, the best customer u to be inserted in the route is chosen as the one, that is non-routed and feasible, for which

$$c_2(i(u^*), u^*, j(u^*)) = optimum[c_2(i_{p-1}, u, i_p)]$$

The customer u^* is then inserted into the route between $i(u^*)$ and $j(u^*)$. When no more customers with feasible insertions can be found, a new route is started by the method, unless of course all customers have been routed.

I1 is one of three insertion heuristics described in (Solomon 1987, page 257), and it is the one which gave the best results. It gives a more precise definition of c_1 and c_2.

$$c_{11}(i, u, j) = d_{iu} + d_{uj} - \mu d_{ij},\ \mu \geq 0$$

$$c_{12}(i, u, j) = b_{ju} - b_j$$

Here b_{ju} is the new time for customer j to be serviced, given that customer u is on the route. The distance between customer i and u is denoted d_{iu}.

$$c_1(i, u, j) = \alpha_1 c_{11}(i, u, j) + \alpha_2 c_{12}(i, u, j),\ \alpha_1 + \alpha_2 = 1$$

$$\alpha_1 \geq 0,\ \alpha_2 \geq 0$$

$$c_2(i(, u, j) = \lambda d_{0u} - c_1(i, u, j),\ \lambda \geq 0$$

This an insertion heuristic that tries to maximize the benefit of servicing a customer trough a partial route being constructed, compared to servicing it on a direct route. By weighting the different parameters, the I1 focuses on different aspects when optimizing the routes. By setting α_2 to zero, the new service times of existing customers is ignored when deciding where a new customer could be inserted in the partial route. The parameter λ, deals with how much the best insertion of an unserviced customer is affected by its distance to the depot.

2.4.2 Improvement heuristics

When an initial solution has been derived, improvement heuristics are employed trying to further ameliorate the routes that have been created.

[1] I1 is constructed with respect to the VRPTW, therefore the need to start and end at the depot

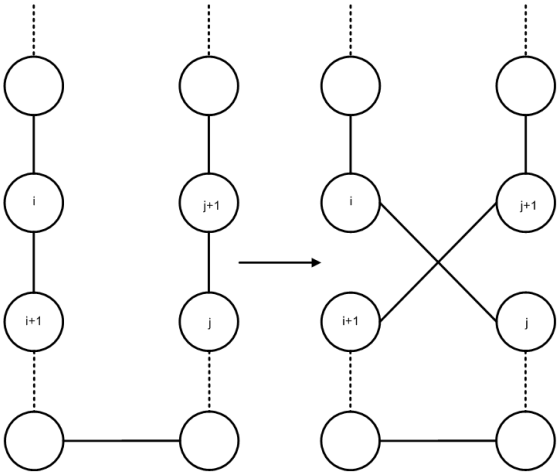

Figure 2.6: 2-opt exchange heuristic

k-opt exchange heuristic (k-opt)

The k-opt exchange heuristic (k-opt) created by Lin (1965), exchanges k arcs in the current routes for k new links. This procedure is done iteratively, until the solution can not be further improved. If N arcs are covered by a route, there are $O(N^k)$ possible ways of selecting the k arcs to be replaced. In Figure 2.6, one iteration of the k-opt is performed, with $k = 2$, to improve an existing route.

2-opt* exchange heuristic (2-opt*)

According to Potvin & Rousseau (1995) the k-opt are not well adapted to problems with time windows, because the orientation of the routes is altered by the exchanges. The 2-opt* exchange heuristic (2-opt*) was introduced to deal with such problems and it also deals with problems of multiple routes. By adding the last customers of a given route at the end of the first customers of another route. The first customers are typically those with early time windows, and the last customers are those having late time windows.

The 2-opt* is useful when replacing arcs of different routes, when dealing with intra-route exchanges the k-opt is to be used. Figure 2.7 shows how the 2-opt* works on two selected routes.

Or-opt exchange heuristic (Or-opt)

This approach is a well known node exchange heuristics . An Or-opt exchange heuristic (Or-opt), considers one, two or three customers connected in sequence in the current

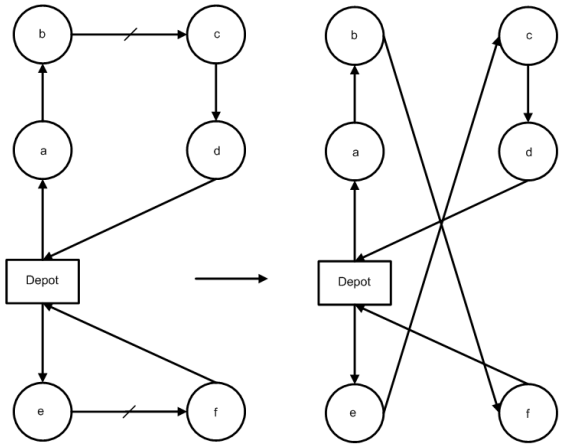

Figure 2.7: 2-opt* exchange heuristic (2-opt*)

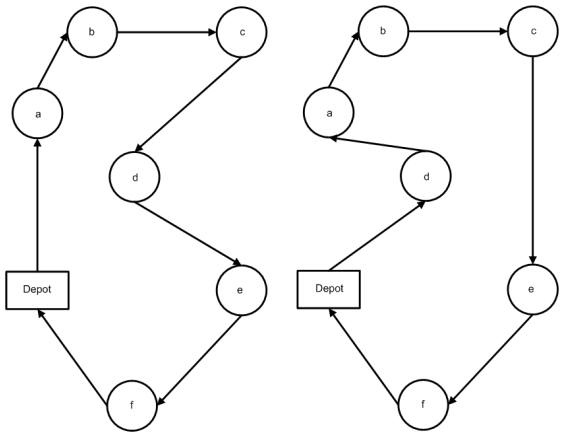

Figure 2.8: Or-opt exchange heuristic (Or-opt)

solution, and tries to improve the solution by inserting the customer(s) at a new location. This heuristic is focused on nodes as compared to the arcs improvement strategies described earlier in this section. Focusing on nodes rather than arcs, does not mean that the Or-opt is completely different than for example the k-opt. The Or-opt are subsets of different 3-opts. Potvin & Rousseau (1995) states that these subsets are more likely to be feasible because only a small sequence of adjacent customers are moved and inserted at a new location which preserves the orientation of the routes. This results in the Or-opt generating solutions that are close to those of the 3-opt for problems with time windows, only in less computation time.

Figure 2.8 shows how the Or-opt may work. The sequence of nodes a-b-c is moved to improve the total distance to be travelled within the initial route. The Or-opt is in addition to inter-route modifications also suitable for intra-route modifications.

2.5 Heuristic frameworks

Heuristic frameworks are combinations of one or more solution methods united in a structured method or strategy to search for the optimal solution of a problem.

2.5.1 Local search

Local search heuristics will try to improve a solution until it reaches a local or global optimum, away from which any single move will worsen the solution. Meta-heuristics work in another fashion in which they allow searching through inferior areas to render possible the discovery of superior solutions. Algorithm 1 shows an overview over how a local search heuristic generally works.

Algorithm 1: Local search

Input: An initial solution x_0, with object function $c(x_0)$
Output: An optimal solution(most likely a locally optimal solution)
$i = 0$
Find all points in neighbourhood $N(x_i)$
Find $x_{i+1} \in N(x_i)$ so that $c(x_{i+1}) < c(x_i)$
while $c(x_{i+1}) < c(x_i) \forall x_k \in N(x_k)$ **do**
    i = i+1
    Find all points in neighbourhood $N(x_i)$
    Find $x_{i+1} \in N(x_k)$ so that $c(x_{i+1}) < c(x_i)$
end
return x_i

The local search often consists of a combination of improving heuristics, put together to find a local optimum relatively quickly. Tightly constrained problems may leave only small areas of the solution space available for the local search to investigate. This and the weakness that the local search has no means available to escape local optima, results in the method only investigating the possibly small part of the solution space left after creating the initial solution.

2.5.2 Multi-start local search

Local search heuristics suffer of the problem of being trapped in local optimum. Martí (2003) argues that some kind of diversification is often needed to allow searching by such methods in alternative neighbourhoods, and overcome local optimums constraining searches in local neighbourhoods. This is especially useful in cases that are closely constrained.

Multi-start local search heuristics rely on diversification of the initial solutions from which the local search starts, to investigate a broader part of the solution space. According to Martí (2003) multi-start methods consist of two phases: first the one where the solution is generated and the second phase where the solution is improved if possible. After iterating through these two steps until some stopping condition is met, the best overall solution is the algorithm's output.

Algorithm 2: Multi-start method

Input: Data to be optimized
Output: The best overall solution (minimal cost or object function)
Initialise $i = 1$
while *Stopping condition not met* **do**
  Construct solution x_i with object function $c(x_i)$
  while *Local search not finished* **do**
    Try to improve x_i
    x_i^{best} is the solution with object function $c(x_i^{best})$
  end
  if $c(x_i^{best}) < c^{best}$ **then**
    $x^{best} = x_i^{best}$
    $c^{best} = c(x_i^{best})$
  end
  $i = i + 1$
end
return x^{best}

Martí (2003) discusses three key classification elements in multi-start methods, when it comes to creation of initial solutions. These are: memory, randomization and degree of rebuild. Memory is based on benefiting from knowledge gathered in previous searches, and using this knowledge when creating new solutions, by incorporating actions that have given good results in the past to a certain degree. Randomization, is a very simple way of achieving diversification. The problem is that one does not control the diversity obtained. Randomization is often combined with deterministic rules, when creating solutions. The degree of rebuild indicates the number of elements fixed from one generation to another. Algorithm 2 is not concerned with the memory element, and it builds each solution from scratch, thus it has no degree of rebuild.

Multi-start heuristics are effective on tightly constrained scheduling problems like vessel scheduling problems, according to (Brønmo et al. 2007, page 904). Two major reasons for this are that the local search neighbourhoods will not allow the searches to search far in the feasible search space, and the relative easy construction of initial solutions. These features are typical features of ship scheduling and routing problems.

2.5.3 Meta-heuristics

In Chapter 2.4.2 improvement heuristics were introduced. These can be employed to search the neighbourhoods of initial solutions to try to ameliorate these. Improvement heuristics have the weakness that they are greedy, and this may lead the search to be trapped in a local optimum. Meta-Heuristics allow non-profitable moves when searching, in order to explore different neighbourhoods, thus reducing the probability of trapping the searching in an inferior area.

Tabu search

The Tabu search uses a local or neighbourhood search procedure to iteratively move from one solution to a new solution in the neighbourhood of the first solution, until some stopping criterion has been satisfied. To explore regions of the search space that would be left unexplored by the local search procedure to escape local optimality, Tabu search modifies the neighbourhood structure of each solution as the search progresses. The solutions admitted in the new neighbourhood, are determined through the use of special memory structures. The search now progresses by iteratively moving from one solution to a new solution in the neighbourhood. To avoid cycling, solutions possessing some attributes of recently explored solutions are temporarily declared tabu or forbidden and added to the tabu list. This unless their cost is less than a so-called aspiration level, see (Cordeau et al. 1997, page 107). Some implementations allow for intermediate infeasible solutions. The neighbourhood where the solution is allowed to move is $N(x_i, TL)$, where TL is the tabu list.

To allow the search to go through an inferior area, short term memory structures are used, namely the tabu list. This list contains solutions that have been visited in the recent past, and these are not allowed in new neighbourhood. The tabu list may also contain structures prohibiting solutions having certain attributes, specific to the problem being solved.

A Reactive Tabu Search (RTS) is a robust search technique that enhances classical Tabu search by allowing the algorithm to automatically adjust the search parameters based on the state and the quality of the search. It allows the algorithm to choose strategies and parameter values at each iteration, instead of using constant parameters or choosing from a predefined set of parameters. The RTS allows the detection of chaotic attractor basins as well as providing a method to escape from such basins and continue the search in new and unexplored areas, (Nanry & Barnes 2000, page 110)

Tabu search is regarded as an effective heuristic for many problems like the VRPTW and the PDPTW. Nanry & Barnes (2000) has used a variant of the search on a m-PDPTW, with good results. It has been regarded as the most efficient approach for a number of

Algorithm 3: Tabu search

Input: An initial solution x_0, with object function $c(x_0)$
Output: The best solution found by the Tabu search
Initiate the tabu list TL
$i = 0$
while *Convergence criterion not met* **do**
  Find all points in neighbourhood $N(x_i, TL)$
  Chose $x_k \in N(x_i, TL)$ so that $c(x_i)$ is minimized without x_i being in TL
  Update the TL
  $i = i + 1$
end
return x_i

problems, see (Brønmo et al. 2007, page 905). For more information on the Tabu search, see Glover & Laguna (1997).

2.5.4 TurboRouter

TurboRouter is a decision support system developed by MARINTEK[2] in collaboration with NTNU[3]. TurboRouter is aimed at helping the planners which have historically done the routing and scheduling manually, not replacing them. The software is developed to have a quick response time, giving high quality results, while developing practical solutions applicable to the real world, and allowing interaction between user and software.

Heuristics are employed to balance finding high quality results, and the need for a quick response time. The different heuristics available are described in Brønmo et al. (2007). A multi-start local search heuristic is used, with a part of each initial solution created in a pseudo random manner to diversify the start solutions. The user has the possibility to specify the frequency of which each neighbourhood is visited.

There are many parameters available for the user to choose from giving a high degree of interaction. This may be different cost structures, cargoes with different priorities, search specifications, and the possibility to manually alter schedules created by the heuristic core of the software.

TurboRouter is unfortunately not modularized, meaning that any attempt to modify the program is difficult. It has been under constant development, and its specification has been changing on the basis of new clients having different needs. This is the reason for the lack of modularity. The documentation of the software was also found inadequate.

[2]Norwegian Marine Technology Research Institute
[3]The Norwegian University of Science and Technology

The software is built on the basis of having orders with an amount of a certain product to be transported from a loading port to an unloading port within possible time windows, and assigning them on a given fleet of vessels. Being commercially available software, it is considered thoroughly tested and relatively bug free.

2.6 Discussion on the different solution methods

There are two major types of solution methods for the type of optimization problem which is being discussed. Complete methods and heuristic methods. Complete methods guarantee the globally optimal solution, while heuristics only can guarantee a relatively good solution (a local optimum).

Both types of solution methods are subject to constant research and amelioration, and with computing speed and power constantly increasing, a variety of new tools are bound to be available in the near future. At the present complete methods have solved problems of relatively small size and complexity, but it is time consuming and unsuitable when dealing with larger problems. Constraints such as time windows and load capacity may rule out a large set of the possible cargo combinations, which is useful in column generation, but this method is still time consuming if it can derive a solution at all.

While complete methods guarantee the global optimum (if solvable), heuristics guarantee local optimum. Local optimum is considered a "trap" by many optimizers, see Martí (2003). Therefore techniques to enable searching a lager part of the solution space, have been developed. The meta-heuristic, Tabu-search and the multi-start local search heuristic are examples of ways to "escape" local optimum. The Tabu-search allows moves that worsen the objective function, while the multi-start method uses different start points to investigate a larger part of the solution space. Effective heuristics are fast compared to complete methods, will always find a solution if the problem is solvable, but it is very probable that the solution found will not be the best solution possible.

Chapter 3

The problem at hand

The problem at hand consists of optimizing crude water-borne oil transport from multiple loading ports, to multiple unloading ports in different regions. Of all the different types of VRPs, this problem most closely resembles the m-PDPTW, seen in Chapter 2.2.4.

This chapter focuses on describing the problem at hand. The complexity of the complete problem is too great for the scope of this report. The important parts of the complete problem at hand will be explained, trying to minimize the use of mathematical notation for reader friendliness motivated by Cordeau & Laporte (2003). A reduced problem to be solved, consisting of the most important features of the real problem is then described. Since the crude oil to be transported is being transported term or company vessels, minimizing cost is the essence. The objective function to be minimized consists of variable costs for the fleet of term vessels available.

3.1 Formulation of the complete problem at hand

Loads in general water-borne transportation will usually have the following attributes:

- loading port
- unloading port
- time windows for loading and unloading
- amount to be transported
- product type
- rate (USD/ton)

In the general PDPTW, all orders have pick-up and delivery nodes defined as seen above. In the problem at hand, this is not the case. There are deliveries to be fulfilled at different ports in different regions. These deliveries are complete with amount, crude grade and time windows in which the oil may be discharged and other properties described below. To fulfil the deliveries, different amounts of the crude oil grades are to be picked up at different ports during pick-up specific time windows, and if necessary several pick-ups may have to be performed to fulfil a delivery. A formulation of the problem at hand is found below. Some assumptions and simplifications have been made on the complete initial problem, in adherence with the scope of the report. These simplifications are clearly stated.

3.1.1 Complete problem description

The problem at hand is defined on a complete graph $G = (\mathcal{N}_A, \mathcal{A}_A)$. The set of all nodes is denoted $\mathcal{N}_A$, and the set of all arcs is denoted as $\mathcal{A}_A$. The nodes are representations of the different ports, indexed by i, j in this problem. $\mathcal{N}_P$ is the set of loading ports (pick-up nodes) and $\mathcal{N}_D$ is the set of discharge ports (delivery nodes). Initially each vessel v is situated in an artificial node denoted by $o(v)$, and at the end of the tour it will be situated at the artificial destination $d(v)$. $\mathcal{N}_A = \mathcal{N}_P \cup \mathcal{N}_D \cup \{o(v)d(v)\}$. Different ports have different characteristics with regards to draft and height restrictions, therefore not all ports are accessible for all vessels. $\mathcal{N}_{Pv}$ is the set of all pick-up nodes $\mathcal{N}_P$ generally available for visit by vessel v, $\mathcal{N}_{Dv}$ is the set of all delivery $\mathcal{N}_D$ nodes generally available for visit by vessel v. Out of all nodes $\mathcal{N}_A$, the nodes available for vessel v is the set $\mathcal{N}_{Av} = \mathcal{N}_{Pv} \cup \mathcal{N}_{Dv} \cup \{o(v)d(v)\}$. In addition to knowing which ports a vessel may visit, there is also the set $\mathcal{V}_{Ni}$ of all vessels $v \in \mathcal{V}_A$ that generally can enter port $i \in \mathcal{N}_A$. Different canals will also pose restrictions on which vessels may enter these, this can be regarded as $\mathcal{A}_{Av}$ which is the set of all the arcs in the model that can be visited by vessel v.

$\mathcal{A}_A$ are all arcs in the model.
$\mathcal{A}_A = \big\{(imjnk) : i \in \mathcal{N}_A, m \in \mathcal{T}_{Wi}, j \in \mathcal{N}_A, n \in \mathcal{T}_{Wj}, k \in \mathcal{K}_{imjn}, (imjnk)exists\big\}$.
$\mathcal{A}_{Av}$ is the set of all arcs vessel $v \in \mathcal{V}_A$ can physically sail.
$\mathcal{A}_v = \big\{(imjnk) : i \in \mathcal{N}_{Av}, m \in \mathcal{T}_{Wi}, j \in \mathcal{N}_{Av}, n \in \mathcal{T}_{Wj}, k \in \mathcal{K}_{imjn}, (imjnk)exists\big\}$

All vessels $v \in \mathcal{V}_A$ are either time/term (company owned) vessels in the set $\mathcal{V}_T$, or spot chartered vessels in the set $\mathcal{V}_S$. This gives the set of all vessels $\mathcal{V}_A = \mathcal{V}_T \cup \mathcal{V}_S$. Note that each spot vessel can be used once during each planning period.

Different grades of crude oil are indexed by c, and $c \in \mathcal{C}_A$ where $\mathcal{C}_A$ is the set of all crude grades.

Q_{cim} is the demand in discharge port $i \in \mathcal{N}_D$, time window $m \in \mathcal{T}_{Wi}$ and crude grade $c \in \mathcal{C}_A$. For pick-up ports ($i \in \mathcal{N}_P$), Q_{cim} is the amount of a certain crude grade to be picked up in the certain time window. An amount q_{imcv} is loaded at port $i \in \mathcal{N}_P$ in time window $m \in \mathcal{T}_{Wi}$ or discharged at port $i \in \mathcal{N}_D$ in time window $m \in \mathcal{T}_{Wi}$. Departing from port $i \in \mathcal{N}_{Av}$ in time window $m \in \mathcal{T}_{Wi}$, l_{imcv} is the load of crude grade c on board vessel $v \in \mathcal{V}_A$.

The problem at hand includes time windows. $\mathcal{T}_{Wi}$ is the set of time windows for port $i \in \mathcal{N}_A$. $\mathcal{T}_{Wi} = \{1, ..., M_i\}$, with M_i being the number of time windows in port i. T_{MNim} is the opening time of time window $m \in \mathcal{T}_{Wi}$ in port $i \in \mathcal{N}_A$. The closing time of time window $m \in \mathcal{T}_{Wi}$ in port $i \in \mathcal{N}_A$ is T_{MXim}. More precisely the opening and closing times of the time windows are denoted as $[T_{MNim}, T_{MXim}] \forall m \in \mathcal{T}_{Wi}$.

$\mathcal{K}_{imjn}$ is the set of routing options between node $i \in \mathcal{N}_A$ in time window $m \in \mathcal{T}_{Wi}$ and node $j \in \mathcal{N}_A$ in time window $n \in \mathcal{T}_{Wj}$ if $(imjn)$ exists. x_{imjnkv} is a Boolean variable. $x_{imjnkv} = 1$ if vessel v sails from port $i \in \mathcal{N}_v$ in time window $m \in \mathcal{T}_{Wi}$ to port $j \in \mathcal{N}_v$ in time window $n \in \mathcal{T}_{Wj}$ via route option $k \in \mathcal{K}_{imjn}$. $x_{imjnkv} = 0$ otherwise.

Due to time restrictions, consisting of time windows and the planning horizon, it's crucial to know the durations of the different actions made by the vessels. Loading/Unloading time is dependent on the amount q_{imcv} to be loaded/unloaded onto/off and vessel $v \in \mathcal{V}_A$ in port $i \in \mathcal{N}_{Pv} \cup \mathcal{N}_{Dv}$ and the load/discharge rate (days/ktons) in that port for crude type $c \in \mathcal{C}_A$. The berthing time in between port i and j is denoted T_{Bij}. This includes deberthing time in port i and berthing time in port j. The time needed to sail leg $(imjnkv) \in \mathcal{A}_{Av}$ for vessel $v \in \mathcal{V}_A$ including berthing time T_{Bij}, is T_{Sijkv}.

The objective function to be minimized is a result of the total cost of the deployment of all term vessels $\mathcal{V}_T$, and potential costs of spot vessels $\mathcal{V}_S$. Fixed costs can be disregarded as it has no influence on finding the optimal routes and schedules, see (Christensen et al. 2007, page 223). The term vessels have three operational modes. These modes are idle, loading/unloading and transit(including berthing and deberthing). The cost structure of the vessel varies with the mode the vessel is in, and is explained more closely in the reduced problem description where it is used in the objective function. C_{PORTiv} is the port fee in port $i \in \mathcal{N}_A$ for vessel $v \in \mathcal{V}_T$. The port cost is a function of the port visited and the size of the vessel, see (Christensen et al. 2007, page 224).

3.2 Formulation of the reduced problem at hand

As stated earlier the complete problem is too complex considering the scope of this report. A reduced problem has been deduced, and this is the problem to be solved. The reduced

problem is less complex than the complete problem, but it has many of the same features and constraints.

3.2.1 Reduced problem description

This problem consists of matching pick-ups and deliveries with the following restrictions, assumptions and objectives. In addition important differences between the real and the reduced problem are stated here.

1. Spot vessels are not allowed, only company owned term vessels are used in this problem.
2. All vessels $v \in \mathcal{V}_T$ are initially without loads, their start time of service and start position may vary from one vessels to the next.
3. The total amount of different crude grades of oil to be delivered at the ports $i \in \mathcal{N}_D$ is equal to the amount to be picked up at the ports $i \in \mathcal{N}_P$.
4. The load on board any vessel $v \in \mathcal{V}_A$ does not, at any time, exceed the vessels weight capacity V_{CAPWv}, or volume capacity V_{CAPVv}.
5. A delivery may be split between several vessels $v \in \mathcal{V}_T$.
6. Several pick-ups can be made by $v \in \mathcal{V}_A$ before making one or more deliveries.
7. One vessel may make several deliveries of different crude grades while on one particular voyage.
8. All pick-ups and deliveries must be performed within certain time windows $[T_{MNim}, T_{MXim}]$.
9. The fleet of vessels $\mathcal{V}_A$ is considered to be heterogeneous.
10. Spot vessels are not used.
11. The pick-up amount may vary in the full problem, in the reduced problem it is fixed.
12. Port costs are excluded from the objective function. Port costs in the complete problem at hand are dependent of the size of vessel used. In the reduced problem the data set only considers vessels of the VLCC type which fall into the same price category, therefore we only look at the fuel costs.
13. Restrictions when it comes to the Suez canal are outside the scope of this report. No vessels will have a choice between going through the Suez canal, and around the south of Africa when necessary. Canal costs are therefore disregarded.

14. The total routing costs of all vessels is minimized.

The total objective function is to be minimized. This is a function of the distance between ports, fuel consumption of the different vessels. In port this vessel has a fuel consumption denoted F_{PORTv} in tons/day. Vessel $v \in \mathcal{V}_T$ has a fuel consumption F_{IDLEv} in tons/day while being idle. The amount of fuel used is dependent on the distance travelled and the vessel v deployed. D_{ijk} is the distance between port $i \in \mathcal{N}_A$ and port $j \in \mathcal{N}_A$ on route option $k \in \mathcal{K}_{imjn}$. C_{FUEL} is the fuel price (kUSD/ton). The cost for sailing between port $i \in \mathcal{N}_A$ and port $j \in \mathcal{N}_A$ via route k for vessel $v \in \mathcal{V}_T$ in kUSD is then derived from some of the variables above. This price $C_{LEGijkv}$ includes pure sailing in ballast, or loaded and costs for berthing and deberthing.

The objective function to be minimized is a result of the total cost of the deployment of all term vessels $\mathcal{V}_T$. The cost of completing the different legs including the port costs is seen bellow:

$$z_1 = \sum_{v \in \mathcal{V}_T} \sum_{(imjnk) \in \mathcal{A}_{Av}} (C_{LEGijkv}) \cdot x_{imjnkv} \tag{3.1}$$

The difference in being idle, and in port loading crude is seen in this equation:

$$z_2 = \sum_{v \in \mathcal{V}_T} \sum_{i \in \mathcal{N}_{Pv}} \sum_{m \in \mathcal{T}_{Wi}} \sum_{c \in \mathcal{C}_{Pi}m} (F_{PORTv} + F_{IDLEv}) \cdot C_{FUEL} \cdot T_{Qicv} \cdot q_{imcv} \tag{3.2}$$

The difference in being idle, and in port unloading crude is seen in this equation:

$$z_3 = \sum_{v \in \mathcal{V}_T} \sum_{i \in \mathcal{N}_{Dv}} \sum_{m \in \mathcal{T}_{Wi}} \sum_{c \in \mathcal{C}_{Di}m} (F_{PORTv} + F_{IDLEv}) \cdot C_{FUEL} \cdot T_{Qicv} \cdot q_{imcv} \tag{3.3}$$

For a vessel that remains idle for the entire period, the cost is as follows:

$$z_4 = \sum_{v \in \mathcal{V}_T} F_{IDLEv} \cdot C_{FUEL} \cdot (t_{d(v)1v} - t_{o(v)1v}) \tag{3.4}$$

The total objective function is:

$$z = min(z_1 + z_2 + z_3 + z_4) \tag{3.5}$$

3.3 Problem assessment and proposed solution method

With regards to the theory on this type of problem and solution methods described in Chapter 2, the most important traits of the reduced problem at hand is discussed. Then, on the basis of the problem traits and the strengths and weaknesses of the different solution methods, a solution strategy is chosen.

3.3.1 Problem traits

The reduced problem at hand, resembles the m-PDPTW, the only major difference being the lack of predefined orders. With the type of goods in the problem at hand being bulk (crude oil of different grades), where the goods are transported from does not matter as long as the correct crude grade is delivered at the right port at the right time. This allows more flexibility when routing the vessels, but it also makes the problem more complex.

Time windows in which the cargo is to be loaded and unloaded are included. These are quite tight restrictions as the different harbours have limited capacity, meaning that any buffer available time wise will be relatively small. Soft time windows discussed in Christensen et al. (2004), allowing time window violation in return of an inconvenience cost is an interesting topic, but outside the scope of this report.

With regards to the potential size of the problem, this is considered to be quite large. The data sets made available suggest a fleet size in excess of 15 vessels, and approximately 50 pick-ups and deliveries. Given that the m-PDPTW is NP-hard, which is verifiable but not solvable in polynomial time, because of its exponentially growing number of solutions with problem size. Any solution method must consider the feasibility of any solution method, in addition to the time aspect.

Summarizing, the problem at hand is very complex, due to its size, the vast amount of variables and restrictions that governs its behaviour. Even after reducing the problem, and decreasing some of its complexity, deriving a good way of solving the problem will be an extensive task.

3.3.2 Proposed solution method

Ideally, finding the global optimum would be the ultimate way of solving any optimization problem, or any type of problem for that manner. Unfortunately today's technology has constraints with regards to computing power and speed. This makes finding global optimum, very time consuming if not impossible. Operations research is focused on finding solutions which are applicable in the real world, where time is an important factor.

After the assessment of the problem at hand, and the possible solution methods, the following judgment has been made: Complete methods are not suitable based on their computation time ,need for computing power, and because the initial goal of this report has been to develop heuristics to solve the reduced problem at hand.

Several heuristic frameworks have been investigated, and among these the reactive Tabu search in Nanry & Barnes (2000) and the multi-start local search presented in Brønmo et al. (2007) both have been thoroughly documented and have shown good results. The reactive Tabu search is quite complicated to implement, and therefore a multi-start seemed to be the best strategy for the purpose of solving the reduced problem at hand.

In addition the reduced problem at hand is relatively complex. Building, implementing and testing a solution that deals with all the aspects of the problem is a large task. With the limited scope of this report, building everything from scratch was not deemed feasible. TurboRouter, a decision support system described in Chapter 2.5.4, is developed and implemented for related problems to the one at hand. It is not modularized, which makes altering its behaviour and further development difficult. Still the benefits were believed to weigh more heavily than the inconveniences. TurboRouters' constructive and multi-start local search heuristics are explained in Chapter 4. The implementation work done to modify TurboRouter in order to solve the problem at hand is described in Chapter 5.

Chapter 4

Heuristic solution method

This chapter describes a multi-start local search heuristic very similar to the one presented in Brønmo et al. (2007). TurboRouter, used in the tramp sceduling problem at hand, has many of these heuristic features implemented. These strategies, or neighbourhoods will be described in this chapter.

To the authors' knowledge there has been limited research on local search based heuristics for ship scheduling problems, outside the work of Brønmo et al. (2007). In general related routing problems like the VRPTW, and the m-PDPTW, several heuristics have been developed. The meta-heuristic Tabu search has also emerged as an effective tool in a number of problems, as seen for instance in Nanry & Barnes (2000).

In Chapter 4.1 the multi-start heuristics method, which is the method to be used in this report, will be briefly explained. This method consists of initial solutions explained in Chapter 2.4.1, and a local search which explores different neighbourhoods explained in Chapter 4.3. Finally in Chapter 4.4, an overview of the total local search heuristic with regards to usage of the different neighbourhoods is given.

4.1 Multi-start with biased random insertion

The multi-start local search heuristic starts by generating several initial solutions. These may be based on some sort of bias random insertion. This meaning that some of the elements in each initial solution are, in a biased random way, inserted into each initial solution. Thereby creating several different start points from which the local search can begin. Each initial solution is subject to a set of different improving heuristics combined in a local search trying to ameliorate the solution, and finding a local optimum. Once a region has been thoroughly explored, the search restarts from a new initial solution.

4.2 Constructive heuristics

Constructive heuristics create initial solutions to optimization problems. These heuristics are to a large extent greedy in the way that they are short sighted in taking the best solution for the current task at hand, not considering the global scope of the problem. Some common constructive heuristics are described in Chapter 2.4.1

4.2.1 Initial solutions of the multi-start local search heuristic

The initial solutions are very important for the performance of multi-start local search heuristics. Initial solutions of high quality, which at the same time are diverse may ease bringing the seach to the optimal solution, or at least close to it. A biased random insertion procedure is used to construct a part of each initial solution, this creates diversity. High quality is brought to the solution by having the rest of the solution constructed by a deterministic insertion heuristic. Combining the biased random insertion procedure and the deterministic insertion heuristic gives initial solutions which after a local search may give good results as seen in Brønmo et al. (2007).

The biased random insertion procedure takes out a percentage of the cargoes, this percentage is predefined. A part of the solution is then created by this set of cargoes. A deterministic insertion heuristic is then used to create the rest of the solution. The heuristic processes every cargo in the list sequentially. Each cargo is assigned to the available ship that gives the highest profit. For further information see (Brønmo et al. 2007, page 905)

4.3 Improvement heuristics

In Brønmo et al. (2007), the local search is split into a quick and an extended part. The reason for splitting the search in to parts is run-time considerations. The run-time considerations in this report are not as important, therefore one extended local search has been chosen.

The local search explores five different neighbourhoods to improve the solution. The search continues until a local optimum is found. The neighbourhoods are of two major types: *Inter-route* and *Intra-route* operators. Intra-route operators are conserned with making improvements on the schedule of one ship, while inter-route operators try to make improvements by moving cargoes between different ships. The different neighbourhoods will be explained in the following sub-chapters.

4.3.1 1-Resequence

The 1-resequence neighbourhood is an intra-route operator. The schedule of ship v is visualized by a string of circles. The different circles represent port nodes. The number of cargoes on the ship is N. Node i represent the loading (pick-up) port for cargo i, while node $N + i$ represents the unloading (delivery) port of cargo i. In Figure 4.1, cargo i is removed from the schedule of ship v, and then re-inserted into the schedule at the best possible place.

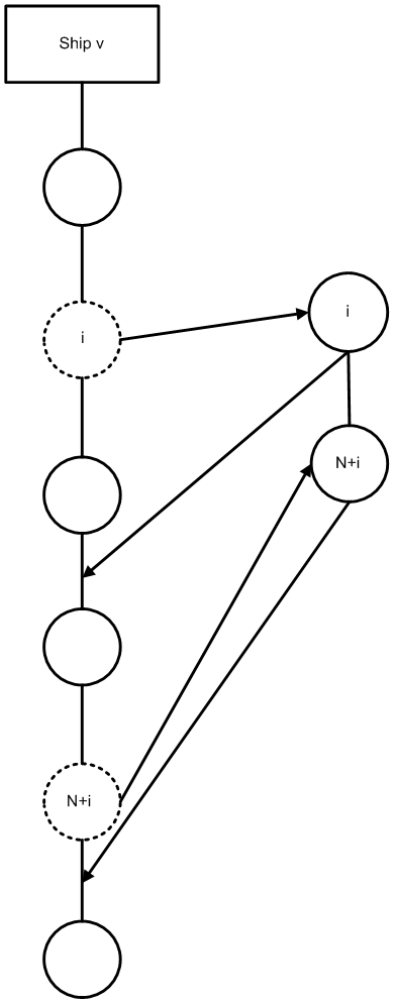

Figure 4.1: 1-resequence neighbourhood, as seen in (Brønmo et al. 2007, page 906)

4.3.2 Reassign

The principles of the reassign neighbourhood, which is also an intra-route operator, is shown in Figure 4.2. Cargo i is removed from the sailing plan of ship v, recalling that cargo i is to be picked up at port i and delivered at port $N + i$. The best insertion into each of the other ships is found, and the cargo is inserted into the ship that gives the best feasible insertion. In this example ship u. If there is one or more cargoes that have been previously rejected (i.e. that have not been possible to assign so far by the heuristic), the reassign operator tries to insert it into the schedule of ship v.

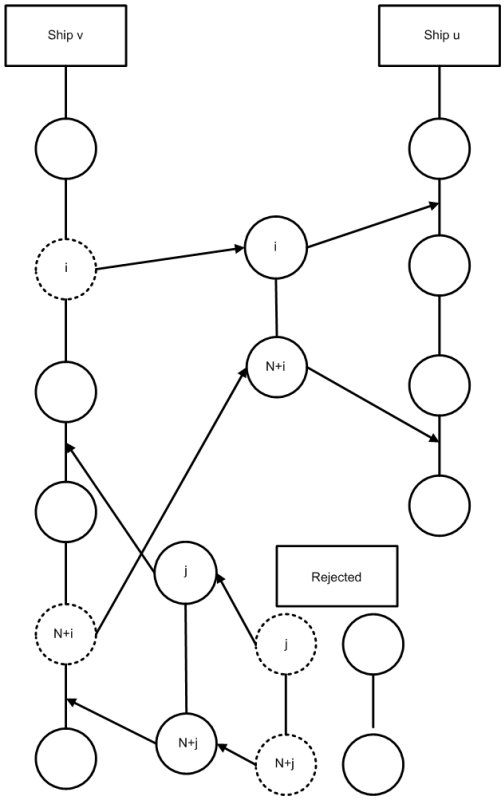

Figure 4.2: Reassign neighbourhood, as seen in (Brønmo et al. 2007, page 907)

4.3.3 2-Interchange

The 2-interchange neighbourhood is an intra-route operator that tries to change one cargo on one ship v, with another cargo on ship u. Figure 4.3 shows how the neighbourhood works. Cargo i is removed from ship v, and cargo j is removed from ship u. Then if possible cargo i is inserted at best position in the schedule of ship u, and cargo j is

inserted in the schedule of ship v.

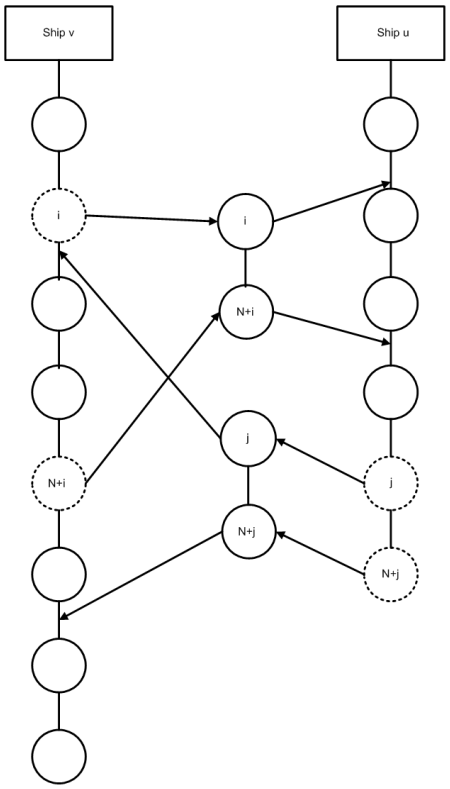

Figure 4.3: 2-interchange neighbourhood, as seen in (Brønmo et al. 2007, page 907)

4.3.4 2-Resequence

2-Resequence is an inter-route operator, and the neighbourhood is shown in Figure 4.4. Cargoes i and j are removed from the schedule of ship v. Firstly cargo i is reinserted at the best possible place on the schedule of ship v, then cargo j is reinserted at the best possible place.

4.3.5 3-Interchange

3-interchange is the same as the 2-interchange neighbourhood. The only difference is that it involves three ships. Cargoes i, j, k are removed from ships u, v, w, in that order. Then cargoes are inserted into the schedules of ships v, w, u, respectively. Figure 4.5 shows how this neighbourhood works.

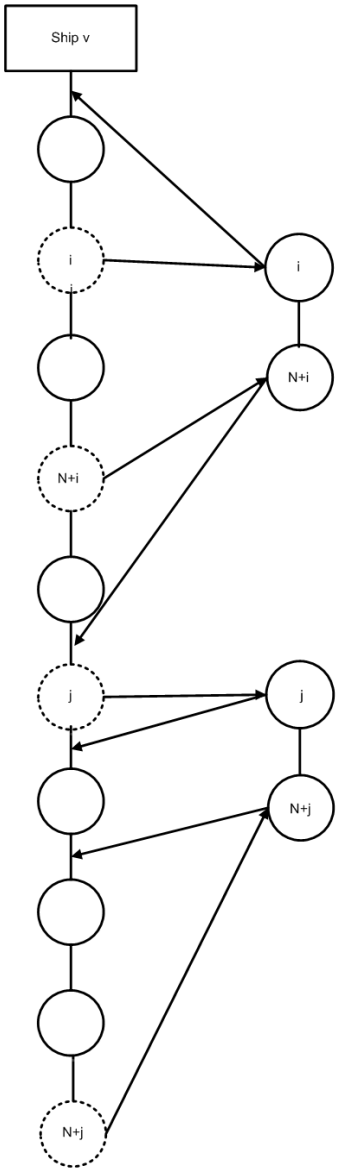

Figure 4.4: 2-resequence neighbourhood, as seen in (Brønmo et al. 2007, page 909)

4.4 Overview of the total improvement heuristic

The different neighbourhoods have differing computational complexities. Therefore Brønmo et al. (2007) argues that only a subset of the neighbourhoods are to be used at each iteration. It is further argued that complex neighbourhoods should not be used as often as simpler ones. This is the way that the local search heuristic used in this report is implemented in TurboRouter.

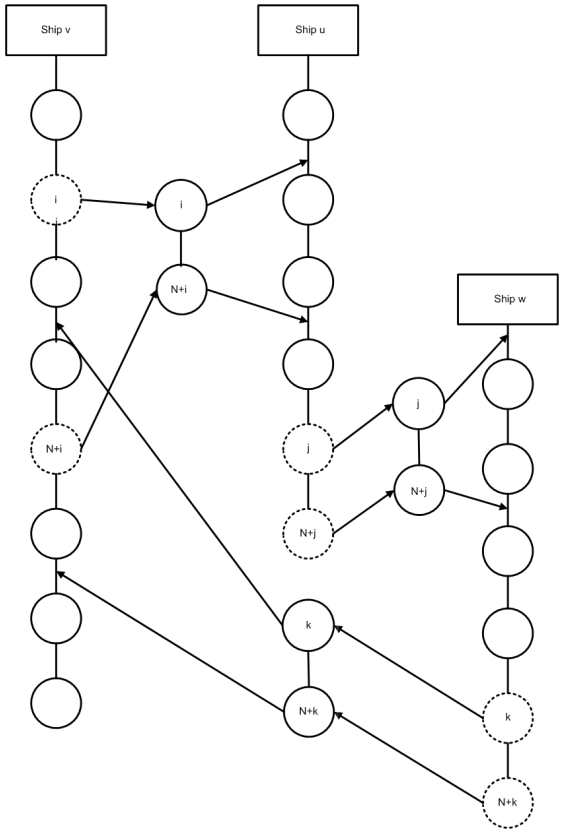

Figure 4.5: 3-interchange neighbourhood, as seen in (Brønmo et al. 2007, page 910)

4.4.1 The flow of the total improvement heuristic

The different heighbourhoods are indexed from 1 to S, S beeing the total number of neighbourhoods available. In this case S would be 5. As the heuristic iterates it uses the different neighbourhoods at iterations specified by a frequency $FREQ_s$,which is the number of iterations between each time neighbourhood s is used, and a first iteration $FIRST_s$, the iteration where the neighbourhood s is first used. For example with $FIRST_s$ being set to 3, and $FREQ_s$ set to 4, the neighbourhood would be used in iterations 3, 7, 11, etc. The test $f(s, iter) = TRUE$? checks which neighbourhoods are allowed, with regards to $FIRST_s$ and $FREQ_s$ and which iteration it is.

If no improvements have been found in an iteration one would normaly assume that a local optimal solution had been found. However this scheme does not investigate every neighbourhood at every iteration. Therefore it is necessary to investigate all the neighbourhoods in S that have not been investigated in this iteration to determine if the current solution is a local optimal solution. If so local optimum is reached, and the local search heuristic terminates.

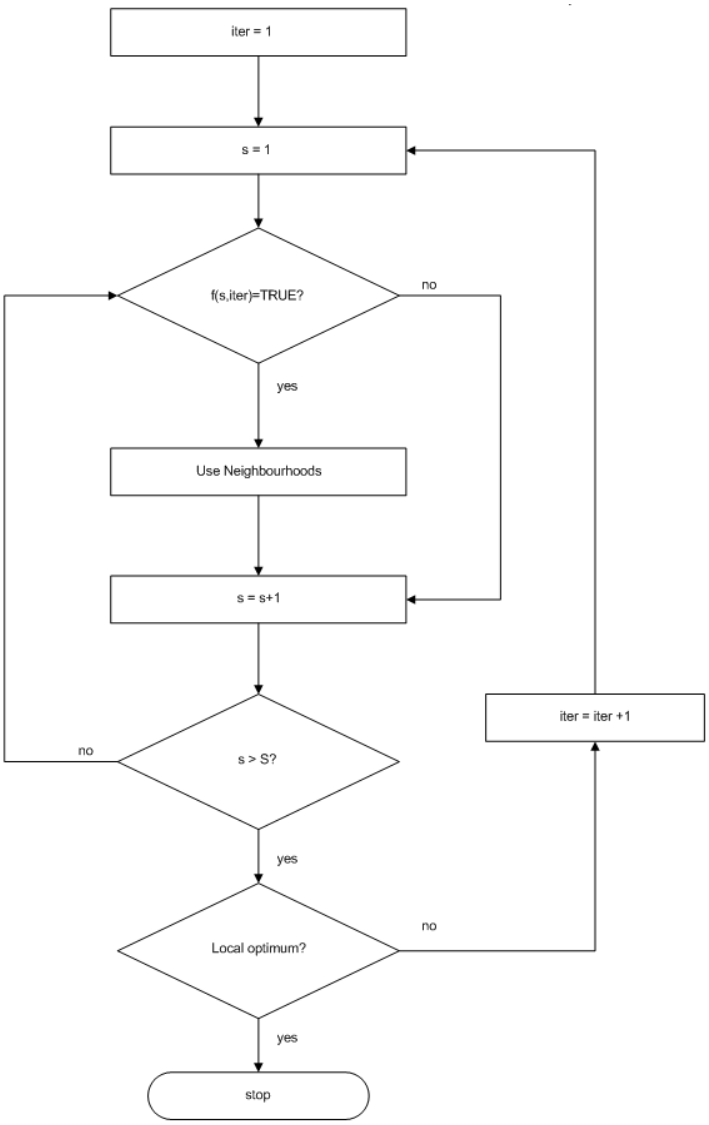

Figure 4.6: Flowchart of the local search heuristic, as seen in (Brønmo et al. 2007, page 908)

Chapter 5

Implementation

This chapter describes the data set and solution strategies used in the implementation behind this report. The goal here is not to describe every detail of the implementation, but give the reader an overview of the work that's been done. Important topics have been more closely visited when found necessary for the readers understanding. Some of the algorithms already implemented in TurboRouter are described in Chapter 4. The data set is first described, before giving an overview of the entire solution. Finally the algorithm matching pick-ups and deliveries is introduced, and pseudo code is to some extent used to describe the different algorithms necessary.

5.1 The data set

Having a data set that is realistic is important in any simulation. In the problem at hand, developing a useful tool for vessel scheduling, the solution needs to be applicable in real life. There is no point in creating an academic solution with no base in reality, or practical usefulness what so ever. This goes for the experimental work as well. Simulating on a dataset that closely resembles actual problems, will give a good indication of how well the solution works.

The problem at hand closely resembles a m-PDPTW, the only difference is that in a m-PDPTW the pick-ups and deliveries are matched as described in Chapter 2.2.4. Matching pick-ups and deliveries is a key topic in this report. The data set is very likely to have several, and most of the time mostly perfectly matched pick-ups and deliveries, easily merged into orders with pick-up and delivery ports, two time windows etc. By "perfect match" being used to describe pick-up and delivery combinations of exactly the same amount of the same product. This because basically, if one has a delivery to fulfil, one usually tries to find one or more pick-ups to fulfil it. This work is done manually in most cases today. If only one single pick-up is available (or can be made available), in

a port near the delivery, to fulfil a delivery. It makes common sense to book this. After matching these "perfect matches" and merging them into orders the problem is like the m-PDPTW. "Perfect matches" are not always the case unfortunately. Sometimes there needs to be performed several pick-ups to fulfil a delivery, and the different pick-up ports may be located in different areas of the world. This makes the matching and merging into orders more complicated. While the difficulty of matching is a result of the availability of pick-ups and deliveries, there needs to be a mass balance of the amount of the different products, or in this case crude grades, to be picked up and delivered.

Time windows, vessels, and draft restrictions also need to be taken into consideration. The mass balance may be in place, but if there is no way that certain deliveries may be serviced by respective pick-ups with regard to time windows, the data set would not have basis in real life. When booking pick-ups and deliveries, and knowing the loading times of the different ports, distances between ports and sailing speed of the different vessels. One would book pick-ups and deliveries with time windows that are compatible. One would also know the draft restrictions of the different ports, and the size of the different vessels. Draft restrictions are excluded from the scope of this report, but this type of restriction is an important factor in routing and scheduling. Distances between the ports are not given exactly, because there is some secrecy surrounding this matter due to safety measures and competition between companies. The approximate location of the ports is known, in addition to the approximate sailing distances. These needn't be exact because the different areas for loading ports and unloading ports are scattered around different parts of the globe.

Summarizing, the data set should contain pick-ups and deliveries where each crude grade is mass balanced, time windows should be compatible, etc. There is also great likelihood of the occurrence of "perfect matches", based on the argumentation above.

5.2 The overall description of the solution

Initially the strategy for solving the problem at hand was to expand TurboRouters functionality, to be able to handle orders consisting only of a pick-up, or a delivery. This would give an extra degree of freedom compared to the order concept already implemented in TurboRouter, with orders being defined as a set of goods to be transported from one place to another. A simplified outline of this model is shown in Figure 5.1. A great deal of work was put into this strategy, but the further down into the software one got, the more complex were the couplings and connections of the orders with the rest of the software. An alternative strategy was therefore sought. Work and ideas on heuristics for this type of problem is described in Chapter 9.

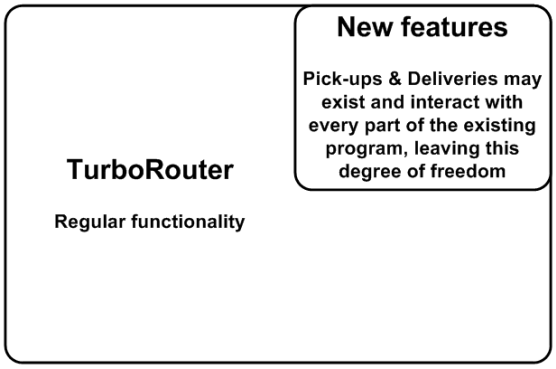

Figure 5.1: Ideal interaction with TurboRouter

With the order concept being a core element in the software, allowing it to remain a core element was the derived solution. This was achieved by pre-matching the pick-ups and deliveries, and thereby merging them into orders by a constructive heuristic, which could be easily integrated into the existing software in TurboRouter. This way of "locking" pick-ups and deliveries together was not optimal, and did not leave any possibility to divide orders into pick-ups or deliveries if the initial merger would turn out to be sub-optimal. One degree of freedom was removed from the solution space, but given the nature of TurboRouter and the scope of this report, there was no real alternative. The implemented and actual model is shown in Figure 5.2.

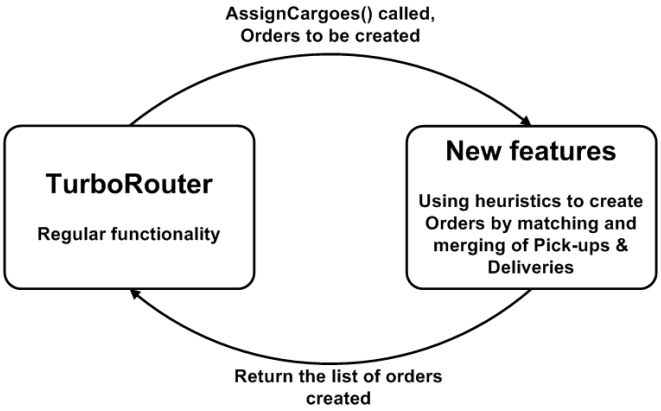

Figure 5.2: Actual interaction with TurboRouter

With regards to the overall flow of the program, it starts by loading the data set. Then, when wanting to assign orders (matching orders and vessels), the matching and merging of pick-ups and deliveries into orders are done. The next step is finding a good allocation of the different orders given the fleet available, minimizing the objective function consisting of the total variable costs of the fleet except port costs. The heuristic being used is

the multi-start local search described in Chapter 4. The multi-start local search heuristic generates a number of initial solutions, of which a chosen number of the best solutions are used as start points for the local search. The selected initial solutions or start points are used iteratively. The local search starts from the first start point, from which it searches the solution space until a local optimum is found. Then the next starting point is used, and once more the local search commences. When every start point has been ameliorated, the best solution found is the one returned. The flow and overview of the total solution is shown in Figure 5.3.

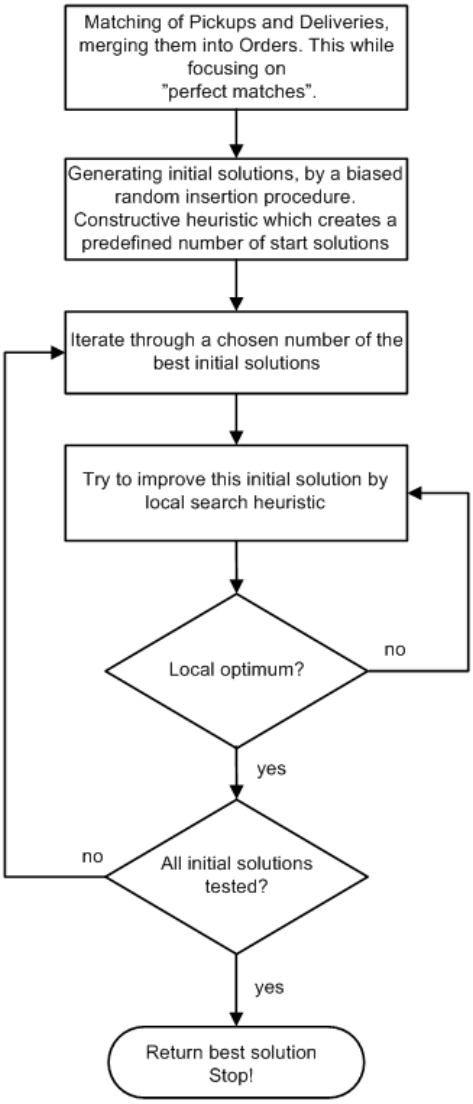

Figure 5.3: Overview of the total solution

5.3 The algorithm for matching deliveries and pick-ups

The first box in Figure 5.3 is described in the rest of this chapter. When matching pick-ups and deliveries and merging them into orders, one degree of freedom is removed, and once the orders are created they can not be split up into their original form. This because the lack of modularity in TurboRouter discussed in Chapter 2.5.4, resulted in the solution strategy seen in Figure 5.2. Therefore it is absolutely crucial that the matching and merging is done in an intelligent way, as the quality of the orders will be likely to influence the objective function of the problem. One way of doing this is to look at how pick-ups and deliveries are booked in the real world and consider the issues that are taken into account when doing this.

On basis of the real world implications described in Chapter 5.1, that will govern the way data sets as the sets used in this report are constructed. Deliveries and pick-ups, to a large extent, will be matched in advance to make sure the different deliveries can be delivered. New pick-ups and deliveries will often be generated on the basis of the deliveries and pick-ups already in place. Algorithm 4 is created with this in mind. Exploiting "perfect matches" is a key focus, as it is stated in Chapter 5.2 that making this solution compatible with TurboRouters already existing software was a necessity.

In addition to focus on "perfect matches", treating the deliveries which have the fewest feasible pick-ups with matching crude grade c, is a principal issue. Leaving deliveries with few, or even one single feasible pick-up to be dealt with at the end of the algorithm turning pick-ups and deliveries into orders, heavily increases the possibility that the feasible pick-ups for these deliveries will be "given" to other deliveries, rendering them unfulfilled. All these issues have been taken into consideration when creating the constructive heuristic, creating the orders.

Based on the amount of feasible pick-ups for each delivery starting with those with the fewest matching possibilities, the existence of perfect matches is sought. If more than one is feasible, the first solution chooses the one with the latest opening time window T_{MNim} that will arrive at the delivery port $i \in \mathcal{N}_D$ with the slowest vessel $v \in \mathcal{V}_A$ including docking, loading and sailing time is chosen. This is done so that pick-ups needed by other deliveries are not taken by deliveries that do not necessarily need them. In the second solution the one with the first opening time window T_{MNim} is chosen. This may leave a greater degree of flexibility time wise for the vessel being assigned this order to pick-up or deliver other loads while on route. The downside is possible unnecessary waiting, or blocking other deliveries from being fulfilled, because the pick-up used was the only possibility for some other delivery. To determine if there was a difference between the two solutions, testing was used, and the solution with the best results in an initial test was considered superior and used in more extensive testing. The tests and results are explained in Chapter 6.

When the order is created as a result of finding a "perfect match", the rest amount of the pick-ups used is set to zero. If no perfect match is available, a set of order combinations associated with the delivery was created. The one consisting of the fewest pick-ups was chosen.

When more than one order combination consisted of the same minimal number of orders, the one with the shortest distance was chosen. Distance here meant that if the minimum number of orders in the order combination was one, the distance was the distance between the pick-up port $i \in \mathcal{N}_D$, and the delivery port $i \in \mathcal{N}_P$. If the order combination consisted of more than one order, distance was chosen to mean the distance between the different pickup ports $i \in \mathcal{N}_D$. This was done because the different pick-ups ports were clustered together in one area, except for one pickup port which was far away from the others. The different pick-ups were sorted in an increasing manner, this so that if many pick-ups were needed, the smallest ones were used in their entirety, so that there were not many small amounts left at different ports for the last deliveries to utilize.

5.3.1 The main algorithm

The outline of the solution is shown in Algorithm 4. The language of the pseudo code is kept as close to that of the solution implemented in TurboRouter, to simplify examination and development of this work. Function calls and variables have kept their function names, when beneficial. Algorithm 4 uses several other methods like for example FindFeasiblePickups(), these are described later in this chapter.

5.3.2 Sorting the deliveries

Deliveries with the earliest closing time windows are chosen to be dealt with first, because these may have fewer possible matching pick-ups. They are therefore sorted by closing time window. This is seen in Algorithm 5.

5.3.3 Finding feasible pick-ups

Finding the feasible pick-ups for each delivery is crucial. This because it is the number of feasible pick-ups that determine which deliveries are matched with one or more pick-ups and merged into one or more orders. The feasible pick-ups of each delivery are determined before any orders are created, therefore one delivery may have fewer pick-ups available when it is matched and merged. This is seen in Algorithm 6.

5.3.4 Finding feasible pick-up combinations

The Algorithm 7, finds combinations of pickups that may satisfy a delivery i. Satisfying a delivery, means that the pick-ups in one combination combined have an equal or lager amount of the right crude grade compared with the delivery. No more pick-ups are added to any combination that has a sufficient amount of crude oil of the right grade.

No combination may exceed 4 pickups. This due to the real world implications of having a large amount of pick-ups considering costs and time constraints. This algorithm does not create the different orders, but continues to add pickups to the vector possiblePick-upCombination until its total amount is sufficient. Order creation is described later on.

5.3.5 Finding possible order combinations

This algorithm is called with a specific delivery as an argument. It deals with creating order combinations out of the feasible pick-up combinations in Algorithm 7, with i called at the very beginning of Algorithm 8. It takes every pick-up combination, and creates corresponding order combinations. These are pick-up combinations, only merged into orders with specific amounts. This because pick-up combinations do not have specific amounts declared, although it is known that they can satisfy the delivery. There needs to be decided how much of each pick-up is to be used when creating potential orders. Having a large number of small leftovers spread around in different ports is not desirable, therefore the strategy here is to use all the smaller pick-ups completely, and leave any leftover from the largest pick-up.

Algorithm 4: CreateOrdersVector()

```
Input: allPickups, allDeliveries, HarbourList
Output: Orders resulting from matching deliveries and pickups
Load allDeliveries and allPickups, SortDeliveries()
FindFeasiblePickups(), max = FindMaxNumberOfFeasiblePickups()
foreach max i do
    foreach delivery j do
        if j has i + 1 feasiblePickups then
            foreach sortedFeasiblePickup k do
                Look for perfect matches
                if perfect match found then
                    perfectMatches ← k
                end
            end
            if numPerfectMatches ≥ 1 then
                foreach perfectMatches l do
                    if perfectMatch is bestPerfectMatch then
                        bestPerfectMatch ← l
                    end
                    if l ≡ numPerfectMatches then
                        new order = bestPerfectMatch + delivery j
                        delivery j → assigned
                        pickup bestPerfectMatch → restAmount= 0
                    end
                end
            end
        end
    end
    foreach delivery m do
        if m notAssigned then
            if m has i + 1 feasiblePickups then
                FindPossibleOrderCombi(m)
                foreach possibleOrderCombi n do
                    Find the possibleOrderCombi consisting of the fewest orders
                    if if num of Combinations consisting of fewest orders > 1 then
                        Find bestOrderCombi with shortest distance
                    end
                    if if num of combinations consisting of min num orders = 1
                    then
                        bestOrderCombi ← this
                    end
                    foreach bestOrderCombi o do
                        new order = [o]
                        m set assigned
                        pickup in [o] → rest− =order.amount()
                    end
                end
            end
        end
    end
end
```

Algorithm 5: SortDeliveries()

```
Input: initialDeliveries
Output: Deliveries sorted by earliest closing time window in vector deliveries
foreach initialDelivery i do
    foreach delivery j do
        if i.unloadStop() ≤ j.unloadStop() then
            noDeliveries = noDeliveries +1
            break
        end
        if j ≡ noDeliveries then
            add i at position j in deliveries
        end
    end
end
```

Algorithm 6: FindFeasiblePickups()

```
Input: allPickups, allDeliveries, allvessels
Output: Finds all the feasible pickups associated with every delivery
foreach delivery i do
    i.GetUnloadStop()
    i.GetCrudeGrade()
    foreach pickup j do
        if pickups crude grade c is a match then
            calculateEarliestArrivalTime()
        end
        if EarliestArrivalTime ≤ unloadStop then
            add j to feasiblePickups for i
        end
    end
end
```

Algorithm 7: FindFeasiblePickupCombinations(j)

```
Input: sortedFeasiblePickups
Output: Finds all feasiblePickupCombinations associated with every delivery i
amount = deliveries[i].getAmount()
foreach sortedFeasiblePickup j do
    if j.getRestAmount() ≡ 0 then
        continue
    end
    if j.getRestAmount() ≥ amount then
        ppc = new PossiblePickupCombination()
        ppc.Add(j) i.Add(ppc)
    else
        totAmount += j.getRestAmount()
        k = j + 1
        foreach sortedFeasiblePickup k do
            if k.getRestAmount() ≡ 0 then
                continue
            end
            if k.getRestAmount() + totAmount ≥ amount then
                ppc = new PossiblePickupCombination()
                ppc.Add(j,k)
                i.Add(ppc)
            else
                totAmount += k.getRestAmount()
                l = k + 1
                foreach sortedFeasiblePickup l do
                    if l.getRestAmount() ≡ 0 then
                        continue
                    end
                    if l.getRestAmount() + totAmount ≥ amount then
                        ppc = new PossiblePickupCombination()
                        ppc.Add(j,k,l)
                        i.Add(ppc)
                    else
                        totAmount += k.getRestAmount()
                        m = l + 1
                        foreach sortedFeasiblePickup m do
                            if m.getRestAmount() ≡ 0 then
                                continue
                            end
                            if m.getRestAmount() + totAmount ≥ amount then
                                ppc = new PossiblePickupCombination()
                                ppc.Add(j,k,l,m)
                                i.Add(ppc)
                            end
                        end
                    end
                end
            end
        end
    end
end
```

Algorithm 8: FindPossibleOrderCombi(i)

```
Input: feasiblePickupCombinations
Output: Finds all the possibleOrderCombinations for delivery i
FindFeasiblePickupCombinations(i)
foreach feasiblePickupCombination j do
    delAmount = i →GetAmount()
    poc = new PossibleOrderCombination()
    if noPickups in Combination ≤ 0 then
        error
    else
        foreach pickupInCombination k do
            pickAmount = k →GetAmount()
            if pickAmount ≥ delAmount then
                amount = delAmount
                delAmount= 0
            else
                amount = pickAmount
                delAmount− = pickAmount
            end
        end
        order = CreateOrder(i, k,amount)
        poc ← Add(order)
    end
    i ← Add(poc)
end
```

Chapter 6

Results

To evaluate the possible pre-matching strategies presented in Chapter 5.3, and the effect of having multiple starting points and/or a local search, extensive testing consisting of 246 computer simulations was completed. 5 sets of optimization problems were constructed from a main data set, and used in testing.

Finding the best pre-matching strategy was the first target, and this strategy was then used in the remanding tests. Secondly, tests to compare the effect fullness of the different modules in the multi-start local search heuristic described in Chapter 4 were employed. To determine the effect of a local search, of multiple start points, and the multi-start local search heuristic 5 tests were created, some having multi-start with varying initial solutions, some having a local search incorporated, and the rest having both multiple start points and a local search.

6.1 Data sets

The data sets were a result of the description in Chapter 5.1. The data set was split into several smaller sets with varying size and complexity, to help identify the behaviour of the heuristics to be tested. The complete data set was also used as one of the sets in testing. Every set had been mass balanced and was possible to match with regards to time windows. Information on the data sets can be seen below.

Set 5 was the complete set of pick-ups and deliveries, and the other sets are sub-sets. The fleet was heterogeneous, and the initial position and starting time was ship specific. The fleet was chosen such that every cargo was serviceable.

Case descriptions	Set 1	Set 2	Set 3	Set 4	Set 5
Planning horizon	53	66	53	61	15
Number of deliveries	18	40	14	42	55
Number of pickups	18	38	12	38	51
Number of vessels	10	15	10	17	19
Number of crude grades	4	11	4	12	15

Table 6.1: Datasets

6.2 Test settings

To investigate the behaviour of the different data sets and the efficiency of the different improving heuristics several tests were created. These tests consisted of combinations of improving heuristics, and multiple initial solutions. Having tests consisting of different heuristics are used to test the effect of the different heuristics. The multi-start local search heuristic with elements of randomization are implemented in TurboRouter, and explained in Chapter 4.

TurboRouter is a decision support system for routing and scheduling software that focuses on optimal vessel allocation on predefined orders. Three main parameters available for optimization are net income, net daily income, and capacity utilization. The goal of this report is to minimize the total costs while ensuring that as many deliveries as possible are allocated to a vessel. Therefore net income is chosen as main parameter.

This is an industrial shipping problem, therefore no income was associated with any delivery. Only the cost of servicing the fleet was calculated. Port costs were disregarded on basis of reduction of the problem at hand in Chapter 3.2. Net income will therefore be negative, and transporting cargo would increase the costs. Given that initial solutions may not contain every cargo to be transported; these would have a better net income than initial solutions with complete allocation of all cargos. The result of this was that even after improving heuristics, some cargoes were left unassigned. This effect was not desirable, and was countered by giving each cargo a lump sum greater than the cost of transporting it. The total lump sum was subtracted from the objective function found by simulation to give correct results.

6.2.1 Settings for the multi-start constructive heuristic during testing

Randomization in the initial solutions of the multi-start local search heuristic results in a great chance that the best solution will vary from one test to another with otherwise identical settings. The amount of randomly inserted elements is denoted as a percentage

of the total number of elements to be inserted. These were chosen without memory and degree of rebuild, key elements described by Martí (2003).

	Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**
Multistart	No	Yes	Yes	Yes	Yes
Number of initial solutions	1	100	1000	100	1000
Number of used initial solutions	1	20	20	0	0
Random percentage of initial solutions	0	15	15	15	15
Local Search	Yes	Yes	Yes	No	No

Table 6.2: Different compositions of algorithms for testing

Multi-start indicates if multiple start points are employed. The number of initial solutions varying from 1-1000 was the number of initial solutions created. Of these initial solutions, a number of initial solutions were used as start points. The best initial solutions were then used. If the test employed the local search, the local search row is marked with a YES.

6.2.2 Settings for the local search heuristic used during testing

The settings used are taken from (Brønmo et al. 2007, page 912). Settings F5, seen in Table 6.3 were chosen there on basis of response time, and quality of results. Recalling Chapter 4.4.1 different neighbourhoods are visited with different frequencies, and first iteration when the neighbourhoods are first visited. Figure 4.6 gives a good overview of how the different neighbourhoods are used in the local search heuristic.

	Parameter settings	
	$FREQ_s$	$FIRST_s$
1-resequence	1	1
Reassign	2	2
2-interchange	5	5
2-resequence	3	3
TryInfeasible	4	4
3-interchange	100	100

Table 6.3: Parameter settings

6.3 Results from testing the different data sets

The test were performed on an Intel Centrino Duo®, 1,73 GHz processor, and 1 GB ram under Windows XP®. The heuristic was developed in C++®. Each test was performed 10 times, due to the partial randomization of initial solutions. The global optimal solutions for each of the sets were not known, therefore the best result found for each set was used as a benchmark against which all other results for this set were measured. The results consist of minimum, average, and CPU time as seen in Figure 6.4. Minimum (Min) is the difference between the best value of this test, and the overall best result for the entire set. Average (Avg) is similar to minimum, only it is the average of the 10 tests compared to the best result. CPU time is the average computation time for the specific test on the specific set.

6.3.1 Finding the best matching heuristic

To determine which strategy was the best, when having two or more "perfect matches" as described in Chapter 5.3, an initial test was performed. Strategy S1 was to use the pick-up with the latest closing time window that was feasible to fulfil the delivery at hand. Strategy S2 was the exact opposite, in stead of taking the "perfect match" with the latest closing time window, the one with the first closing time window was chosen. Both cases were tested on the same data set, namely Set 5 described above. This set was used because the slight difference between S1 and S2 had an effect on the matching for this set, but not for sets 1, 2 and 3. All results were compared the solution with the best objective function found. It can be noted that Test 1 consists of only one starting point, therefore the Avg. gap (%) has been omitted from this test.

	Test 1			**Test 2**			**Test 3**		
	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)
S1	8.55	-	111	0	4.18	3896	1.32	3.17	4250
S2	9.26	-	96	0.48	1.63	5568	0.098	1.98	4641

Table 6.4: Results from comparing pre-matching heuristics, part 1

Strategy S1 gave the solution with the best objective value, but the overall results of Strategy S2 were superior, hence it was chosen as the strategy used in the remaining tests. S2 also had a longer computation

	Test 4			Test 5		
	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)
S1	10.85	16.36	135	8.66	12.10	1070
S2	5.87	8.72	141	3.71	6.18	1088

Table 6.5: Results from comparing pre-matching heuristics, part 2

6.3.2 Testing with the best matching heuristic

With the pre-matching heuristics tested, the best one was used to determine the efficiency of a local search, multiple start points, or a combination of the two in the multi-start local search.

	Test 1			Test 2			Test 3		
	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)
Set 1	13.68	-	9	0.01	2.91	4112	0	3.95	765
Set 2	44.75	-	61	6.14	10.66	2954	0	9.12	1931
Set 3	0	-	1	0	0	31	0	0	91
Set 4	4.16	-	64	0	4.11	1218	1.67	3.05	1664
Set 5	9.15	-	96	0.38	1.53	5568	0	1.88	4642

Table 6.6: Results from testing, part 1

	Test 4			Test 5		
	Min. gap(%)	Avg. gap(%)	CPU (s)	Min. gap(%)	Avg. gap(%)	CPU (s)
Set 1	0	6.43	30	0	3.95	257
Set 2	25.72	28.96	76	17.44	23.03	556
Set 3	0	0	9	0	0	71
Set 4	2.61	7.12	81	1.86	4.31	643
Set 5	5.77	8.61	141	3.61	6.08	1087

Table 6.7: Results from testing, part 2

With regards to the different sets results varied greatly, both in terms of computation time and gap from best solution found. In Set 3, the same solution was found every time. Compared to Set 2 with a gap of 44.75 % from the result in test 1 by a single-start local search, compared to the best solution found by the multi-start local search in test 3. The best solution found for Set 1 was found in three tests, namely Tests 3, 4 and 5. Computation times varied greatly, from 1 second to 5568 for the largest set being tested

by Test 2.

	Best solution	Worst solution	Range response time
Test 1	0	3	1-61
Test 2	1	0	31-5568
Test 3	2	0	91-4642
Test 4	0	1	9-141
Test 5	0	0	71-1087
Tie between Test 3-5	1	-	-
Tie between all tests	1	-	-

Table 6.8: Overview of results

Regarding the quality of the heuristics, Test 3 was the largest multi-start local search heuristic, and it found 4 out of 5 of the best solutions although 2 of these were ties. In the other end of the scale was the single-start local search heuristic in Test 1, finding the "worst" solution in 3 of 5 sets. One may also note that for Set 3 the best solution was found in all tests. Thereby actually leaving 4 possible "worst" solutions. At least one of the two tests using multi-start local search heuristics was among those finding the best solution for each set. Tests 4 and 5, consisting of multiple initial solutions found the best solution on two occasions. One time tied with Test 3, and the other time all tests found the same solution.

Considering the time aspect of the different tests, Test 1, a single-start local search heuristic was the fastest, followed by Test 4 which created 100 initial solutions and returning the best one. Tests 2 and 3, both multi-start local search heuristics, were the most time consuming, with Test 2 using an average of 5568 seconds to solve Set 5. In addition the tests without local search (Test 4 and 5), had very predictable computation times within 3 % of the average computation time. For the multi-start local search on the other hand, computation times varied much more. In the most extreme case the lowest computation time was 1/3 of the average computation time for that specific Test/Set combination. The complete results can be reviewed on the enclosed compact disc.

Chapter 7

Discussion

7.1 The Problem

Optimization of water-borne crude oil transport was the topic of this report. Maritime transportation is a field with observed significant growth in research according to Christensen et al. (2007). The studied problem was an industrial shipping problem, and a great deal of literature is available for similar problems. Industrial shipping problems focus on minimizing the expenses, therefore net income was chosen as the objective function.

There is a significant difference between the problem at hand and traditional water-borne transport optimization problems. Traditionally optimization of maritime transportation has been focused on finding the best allocation of a given set of cargoes on a given fleet, and in some cases the fleet can be extended by the use of available tramp ships. In addition each cargo traditionally has a specified loading port and unloading port. In the studied problem on the other hand, there were decoupled pick-ups and deliveries, needing to be matched as opposed to cargoes in the traditional sense.

7.2 Strategic choices made

Optimization problems like the one at hand are very complex, and there is a multitude of factors to be considered. Creating heuristics applicable to real world problems is the objective of this report. The problem was therefore reduced, and factors that were considered of more academic than practical sort were disregarded. Still after reducing the problem, the task ahead seemed massive considering the scope of this report.

Object oriented programming languages are well suited for problems like the one at hand, due to the large amount of entities (ships, ports, pick-ups, deliveries) in the problem.

Therefore the solution was decided to be programmed in an object oriented programming language. There were three alternatives of how to implement the solution: building everything from scratch, reusing some code from TurboRouter and building the rest of the necessary structures, or further development of TurboRouter. TurboRouter is a decision support system, developed for cargo based traditional water-borne optimization problems.

Ideally one would build the solution from scratch, with a specification fit to serve the demands of the problem. Due to the problem complexity combined with the relatively small amount of object oriented programming experience of the author, this strategy was considered to big a task. TurboRouters lack of modularity and documentation left reuse of code a complicated solution strategy. Some code and certainly many of the basic data structures could have been copied and used, while leaving freedom to develop a program for the specific task ahead. The other real alternative was using TurboRouter as it was and modifying it to fit the requirements of the problem at hand. This with the known lack of modularity and documentation, but also with most of the structures needed already implemented and tested.

Development of TurboRouters already existing code was chosen because it was believed that it was possible to modify the software to successfully deal with the problem a hand in spite of the lack of modularity. Therefore using TurboRouters already existing software was thought to result in less work having to be spent constructing the basic functionality of the program, and more effort could be spent developing heuristics to solve the problem at hand. Being a commercially available software package, the need for testing of existing software for bugs and errors would also be minimal. Testing and verification is normally a quite time consuming and complicated task. Reusing parts of the existing code in TurboRouter, was considered a good alternative, but it was believed that construction of basic functionality would be to demanding. Such a solution would also need extensive testing.

7.2.1 Advantages and disadvantages

By deciding to further develop TurboRouter, many of the necessary tools were in place, reading from file was available with little modification, sailing plans, and a graphical user interface. The main challenge was modifying it to make it able to deal with the problem at hand. After extensive and tireless effort over a significant period of time this plan was abandoned due to its lack of modularity. The order/cargo structure was linked to mostly every other part of the software, and removing or changing it would mean that most of the program had to be changed. It was simply not possible to modify the order structure imbedded in the software, consisting of traditional cargoes with loading and unloading ports, to deal with pick-ups and deliveries that were not matched, within the scope of

the report.

With the deadline of the report closing in, an alternative solution was sought. Because the orders/cargoes are such integral parts of the software, they could not be removed. Therefore a solution based on keeping and using these was derived. The problem at hand still consisted of pick-ups and deliveries which were not matched. In order to use TurboRouters existing functionality, orders needed to be created. By utilizing a pre-matching heuristic, the pick-ups and deliveries could be matched and merged into orders. The orders could then be passed to the already existing structures of TurboRouter, and a similar heuristic framework to the one presented by Brønmo et al. (2007) was used to find a good allocation of the orders on a given fleet.

This final way of implementing the solution was not optimal. The pre-matching resulted in a loss of one degree of freedom when constructing heuristics, because no order could be split into pick-up and delivery once it has been created within the existing functionality of TurboRouter. This property of having separate pick-ups and deliveries was an important part of the problem at hand, and would have made possible a whole new class of neighbourhoods to be investigated. Ideas and background for further work on such neighbourhoods is presented and discussed in Chapter 9.

7.3 Heuristics used

The objective of this report was to construct, implement and test different heuristics for the problem at hand. Complete methods were also investigated to some extent, but mainly to identify strengths and weaknesses of such methods. Two major frameworks were identified. Improving heuristics and more precisely a local search or a multi-start local search, and meta-heuristics like the Tabu-search. Improving heuristics improve a given solution until a local optimum is found, while a Tabu-search allows moves that temporarily degrade the solution, in order to escape local optimum. Both the Tabu search and the local search are dependent on an initial solution from which the search can begin. Normally some sort of greedy constructive heuristic is used to generate an initial solution. Neither a local search nor a Tabu search may return a "worse" solution than its initial solution, but often an improved solution is returned.

Multiple start points is a strategy employed by Brønmo et al. (2007) and Martí (2003) in order to investigate a broader part of the solution space. The pseudo randomization in the constructive heuristic generates a set of starting points for a local search, thus increasing the probability of finding several optimums of which the best one is chosen. A multi-start local search framework has already been implemented in TurboRouter and was used in testing, in addition to the pre-matching heuristic constructing orders from pick-ups and deliveries.

The pre-matching was constructed on the basis of the real world implications of booking pick-ups and deliveries. Existence of "perfect matches"(pick-ups and deliveries of matching goods and times) was a result of these implications. This trait was at the centre of the pre-matching, in addition to making the orders as cost effective as possible, and matching the deliveries which had the fewest pick-ups to service them first. Making them cost effective was done by using as few pick-ups to fulfil any delivery as possible, and making sure that if more than one pick-up was to be used they were at ports as close together as possible. Two different pre-matching strategies were created and tests were performed to determine which one was the better for the problem at hand. Pre-matching could have been done in a variety of ways. The deliveries could have been sorted by amount to be delivered, based on time windows, crude type etc. The strategies used were chosen based on the focus it had on the nature of the problem to be solved.

7.3.1 Advantages and disadvantages

Choosing a heuristic solution method means that finding the global optimal solution is considered less important than finding a good solution relatively quickly, or that a generating a solution with a complete method is not possible for the problem at hand. Complete methods are focused on finding the global optimal solution, but in addition to being very time consuming, such methods may not work on large problems.

The Tabu search is relatively difficult and complicated to implement, but it allows searches in a broader part of the solution space and is effective for the PDPTW according to Nanry & Barnes (2000). Improving heuristics finish searching when they reach a local optimum. Creating multiple starting points is a way to ensure that a larger part of the solution space is investigated.

The two pre-matching heuristics were quite similar, only separated by how they allocated multiple "perfect matches". The first one focused on choosing the pick-up with the latest opening time window, this was done not to ruin opportunities for deliveries nor yet allocated. The other strategy has the exact opposite strategy and chooses the pick-up with the earliest opening time window. This strategy can give more flexibility with regards to performing other tasks in between loading and delivering this goods.

7.4 Results

Before discussing the results, it should be noted that the optimal solutions of the different data-sets were not known. The results of the different tests were therefore compared to each other within each data set. This to determine their relative behaviour, and the quality of their solutions.

The two strategies proposed used in pre-matching they were tested on Set 5. Strategy S2 gave the best results, giving an indication that time wise matching and merging pick-ups and deliveries that are a bit separated in time give better results than having them close together. This is believed to be a result of this strategy leaving more flexibility for the ship having been assigned the given cargo to perform other pick-ups or deliveries in between loading and unloading of the cargo in question. Further evidence for this increased flexibility and with that a broader solution space is the higher computation times for this strategy seen in Chapter 6.3.1.

Testing the effects of the multi-start local search heuristic was done by comparing it to a single-start local search heuristic, and multiple initial solutions heuristics. The multi-start local search was superior compared to the other heuristics, and especially the single-start local search which found the worst solution in 3 out of 4 data sets. In Set 3 all heuristics found the same solution. Port and canal costs are not considered, therefore the differences in percentages are larger than they may have been with these costs included.

Comparing results from tests 4 and 5, it is not surprising to see that the higher number of initial solutions created when a part of that solution is randomized, the better the best initial solution found normally is. This property is not so clear when considering tests 2 and 3. Remembering that Test 2 created 100 initial solutions, and used the 20 best solutions as starting points for a local search. For Test 3 the respective numbers were 1000 and 20. Test 3 had slightly better results, but the difference was not large. In addition it is interesting to see that Test 2 has higher computation times for 3 of the sets. Because the computation time of the constructive heuristic is close to linearly dependent on the number of initial solutions in the respective data set, the local search must be the reason for this unexpected result with regards to computation time. One possible explanation can be that generating a large amount of initial solutions in a relatively small set. This may have lead to many similar solutions among the best 20 solutions which in turn may have been start points leading to quick local searches.

Set 1 and 3 give the perception of being very similar. They have an identical fleet available, the same number of crude grades, and a relatively similar number of pick-ups and deliveries. Their computational results are very different. Set 1 had relatively high computation times and varying results in the different tests. Set 3 on the other hand had the lowest computation time in every test, and the same solution was found by every test performed. Looking more closely into the data set an important difference is found. Set 1 consists of many small pick-ups and deliveries, resulting in a great number of possible combinations, while Set 3 had more large pick-ups and deliveries. In addition a great deal of the pick-ups and deliveries in Set 3 had only one or two possible vessels to be transported by. These characteristics greatly reduced the number of possible combinations which in turn reduced the solution space in which the constructive heuristic and

local search could operate. A smaller area in which to search would mean quicker searches.

7.5 Real world applicability

With regards to real world applicability computation times are essential. Having a worst case computation time of nearly two hours and an average of one and a half hour for the largest set is to long even though it can be reduced by using a more powerful computer. In addition it may be noted that the used existing code in TurboRouter, and the additional code added has not been focused on reducing arithmetic operations, and there is a bit of redundancy slowing down the computation. Implementation in an object oriented programming language like C++ will also result in the software spending some of its computation time creating and destroying objects.

Time is of the essence in most real life applications of optimization tools. Response time of the different tests will vary for every set containing a degree of randomization, this because constructive and especially improving heuristics depend greatly on their initial conditions. A great deal of variation was seen in the tests consisting of multi-start local search. This variation was due to the local search having a computational time depending on the start point. One start point may lead to a time consuming local search, while another may for example be a local optimum. The variation in computation time is an important feature to be aware of if implementing such methods in real life applications. Because knowing the run time of a simulation may be important. One other reason behind the high computation time may be excessive fleet size used in the data sets which broadened the solution space. The relation between the size of the solution space and the computation time was evident regarding the computation times for Set 1 and Set 3.

The heuristic framework used in this assignment incorporates a large number of neighbourhoods, and to determine the computation time of each neighbourhood they should be tested. If the total computation time needs to be reduced some of the more computational expensive neighbourhoods could be removed, or visited less frequently. In addition the number of start points could be reduced, and a quick and extended local search as presented by Brønmo et al. (2007) can be used. One single local search, or constructing multiple initial solutions were much quicker heuristics, but again they generally delivered lower quality solutions

Chapter 8

Conclusion

A ship scheduling and routing problem in optimization of water-borne crude oil transportation has been investigated, and presented in relation to carefully selected background literature. On this basis a proposed heuristic has been developed, and implemented supplementing structures already available in TurboRouter. A multi-start local search with pre-matching of pick-ups and deliveries heuristic was chosen based on an assessment of problem size, problem type, real life applicability and existing software. The heuristic has been subject to rigorous testing, and the results have been carefully examined, presented and discussed.

Two strategies have been developed for pre-matching and then tested. Strategy S2 gave the best results, giving an indication that time wise matching and merging pick-ups and deliveries that are a bit separated in time give better results than having them close together.

Testing the effects of the multi-start local search heuristic was done by comparing it with a single-start local search heuristic, and a multiple initial solutions heuristics. The multi-start local search generated superior solutions compared to the other heuristics, but it is computationally too expensive and therefore removing or less frequently visiting some neighbourhoods should be considered if this framework is to be real world applicable. The data sets may also have been too flexible because of the excessive fleet available. This flexibility would lead to a large solution space, in which searching can be time consuming.

Pre-matching of pick-ups and deliveries and then using constructive and improving heuristics was not the ideal way of solving the problem. This choice was a result of its size and complexity. It is recommended to build a solution from scratch, possibly using some of the structures TurboRouter, to investigate heuristics created specifically for problems of this kind. Ideas not implemented in this report and further work is presented in Chapter 9.

Chapter 9

Further Work

Because of the difficulties encountered when trying to modify TurboRouter, some ideas and work was not implemented and tested. They are instead described here, in case of further development on problems like the problem at hand. Maintaining the degree of freedom, provided by pick-ups and deliveries not being merged into orders, would provide more possibilities when it comes to both constructive heuristics and improvement heuristics.

The multi-start local search framework presented in Chapter 4, is used as a starting point for the search for new heuristics. Improvements and alternative heuristics to those presented in that chapter are introduced below.

9.1 New constructive heuristics

The constructive heuristic used in this report deals with assigning orders already created. This means that even the global optimal allocation of these orders, may still possibly give a suboptimal solution because the orders created may not have been created in an optimal manner. Remembering that global optimum and complete methods are not the focus of this report, it is still a goal to create the best heuristics possible.

In Brønmo et al. (2007) the list of unassigned cargoes is first sorted increasingly by the start of the pick-up time window, or decreasingly by cargo quantity. In the constructive heuristic used in the main part of this report the cargoes, or orders being the term used here, are only sorted by start of time windows. But one is free to sort by any variable one likes in this form of constructive heuristic. After this has been done each of the cargoes or orders are assigned to the available vessel that gives the highest profit

It would be interesting to test a similar heuristic, which not only matches order and vessel in a greedy manner, but matches a delivery with the pick-up(s) and available vessel that gives the highest profit. This manner of matching is done considering the fleet available, and not blindly like in the constructive heuristic described in Chapter 4.

Algorithm 9: New constructive heuristic

Input: pickupList, deliveryList, shipList
Output: An initial solution
Sort the deliveryList by one or more criteria
while *not iterated through all deliveries* **do**
  Get next sorted delivery from deliveryList
  match delivery i with the pick-up(s) and vessel that gives the highest profit
end
return solution

Algorithm 9 shows an outline of a proposed constructive heuristic. Placing it in a multi-start framework would be a good idea, and the start points should be based on both different sorting criteria, and a degree of randomization.

9.2 New improvement heuristics

Having the extra degree of freedom also leaves new possibilities when it comes to creating effective heuristics. The ones proposed in Brønmo et al. (2007), are focused on orders with pick-up ports and delivery ports. Therefore in order to move an order from one vessel to another, both pick-up and delivery node corresponding to that order must be removed from the vessels sailing plan. This can be seen in Figure 4.3 for example, where orders i and j, represented by pick-up nodes i and j and delivery nodes $N+i$ and $N+j$ are interchanged between two vessels.

9.2.1 Neighbourhoods

Recalling Chapter 4.3, the neighbourhoods were categorized as either *Inter-route* or *Intra-route* operations. There may not be a great deal to improve in the intra-route operations. In the 1-resequence neighbourhood, both pick-up and delivery node are removed and re-inserted. This may alternatively be done by resequencing the pick-up nodes first, and then the delivery nodes. This because transportation of water-borne crude oil, normally consists of loading the vessel at one or more ports, then sailing to one or more ports to deliver the crude oil before returning for more pick-ups. The downside of this way of

inter-route operation is that the optimal sequencing of the pick-up nodes, only considering the pick-up nodes, may give a poor solution when the delivery nodes are added. Testing will give an indication of the performance of this neighbourhood, but the author is not overly optimistic.

The potential of inter-route operations seems greater. Here the possibility of only interchanging two pick-up nodes or two delivery nodes is possible, in stead of interchanging both pick-up and delivery node at the same time.

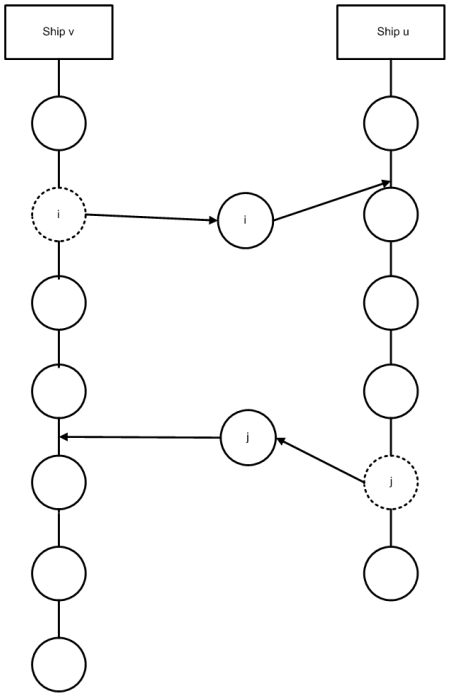

Figure 9.1: New 2-interchange neighbourhood

Figure 9.1 shows an interchange neighbourhood where nodes i and j may both represent either two pick-up nodes, or two delivery nodes. This extra degree of freedom offers a great deal of flexibility optimization wise, and several other neighbourhoods may be constructed with this in mind. Especially the 3-interchange neighbourhood may use this freedom. There are of course limitations on these neighbourhoods. Size and crude grade of the nodes to be interchanged must match, and time constraints must be met. Often, and in the complete problem at hand, there is some slack when it comes to the amount to be picked up. Often in the $+-10$ % range. This increases probability of finding matches that are interchangeable.

9.2.2 The Suez Canal

The Suez Canal is an interesting topic with regards to water-borne transportation of crude oil. This canal is heavily priced. It also poses draft and size restrictions on the vessels wanting to use it, which means that it can not handle the two largest classes of crude carriers which are above the size limit of 200.000 dwt.

The Sumed pipeline is a 320 km long oil pipeline in Egypt, which runs from Ain Sukhna terminal on the Gulf of Suez to Sidi Kerir on the Mediterranean. It provides an alternative to the Suez Canal for transporting oil from the Persian Gulf region to the Mediterranean. The Suez Canal and the Sumed pipeline together create a quite complex environment in which to create effective optimization tools. Not passing through the canal, would yield substantial economic benefits from avoiding the canal costs, but also the economic benefits of being able to use larger vessels. The Suezmax is the largest crude carrier capable of accessing the canal, leaving the two largest classes VLCC (those above 200.00 dwt) and ULCC unable to use the canal.

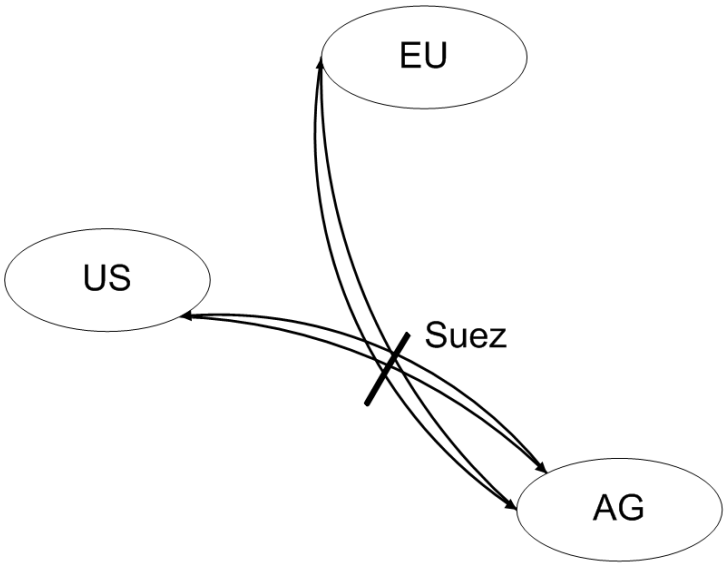

Figure 9.2: Transportation using the Suez Canal

Figure 9.2 shows water-borne transportation using the Suez Canal. AG stands for the Arabian Gulf which is the main area for loading cargo in the problem at hand, but may represent any port in the Indian or Pacific Ocean transporting goods through the Suez Canal. US means the United States of America, and EU means Europe.

An alternative way of transporting crude oil is shown in Figure 9.3. Here the oil is transported by pipeline from Ain Sukhna to Sidi Kerir. Sumed may be considered as two

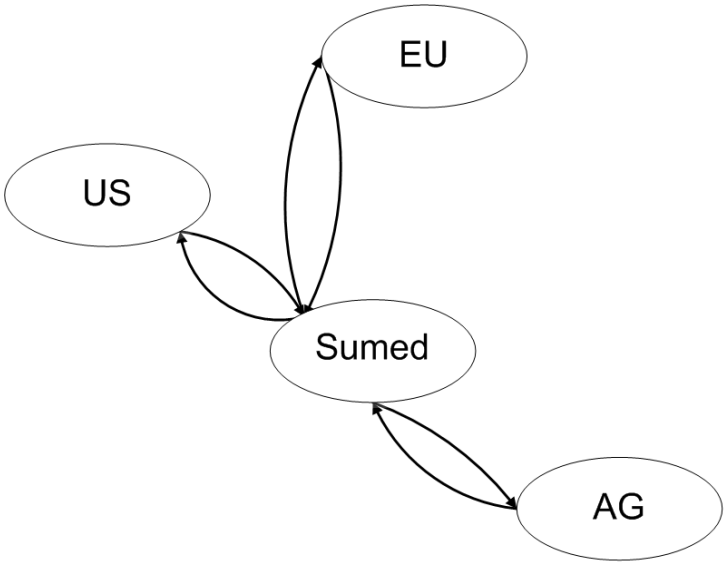

Figure 9.3: Transportation using the Sumed pipeline

ports, Ain Sukhna for unloading, and Sidi Kerir for loading. There is a time delay from when the oil is unloaded, and when it is available for loading. This due to the length of the pipeline.

Considering fleet size and mix are considered strategic problems see Christensen et al. (2007), and are therefore not discussed in great detail here. But one possibility is having a set of vessels only occupied with transporting oil from Sidi Kerir to the markets in Europe and North-America, and another set of vessels transporting oil from the Arabian Gulf to Ain Sukhna. A combination of having a few designated vessels on both sides of the canal and some that pass through it from time to time may also be possible. These ideas can be tested in a type of constructive heuristic, where some vessels can be set to operate each side of the canal. A combined version may also be tested.

On a more strategic level testing the potential of these solutions, should also incorporate pick-up and delivery times aimed at facilitating these routes, as sailing times are known. In order to make such a solution possible and well functioning, pick-ups and deliveries must be well timed. Having a relatively long time horizon, booking "good" time windows for this purpose should be a feasible task.

With regards to improving heuristics created on the basis of the existence Sumed pipeline there is a potential neighbourhood consisting of rest of voyage exchange, which can be investigated. If two vessels are to approach the Suez Canal around the same period, one from the Mediterranean side, the other from the Gulf of Suez an interesting opportunity

arises. In this situation with a similar problem to the one at hand in this report, any vessel approaching the Suez Canal would be empty (except when transporting crude oil from West Africa). Vessels on route towards the Suez Canal arriving at the Gulf of Suez would be loaded with crude oil. These vessels may exchange their next pick-up and delivery nodes.

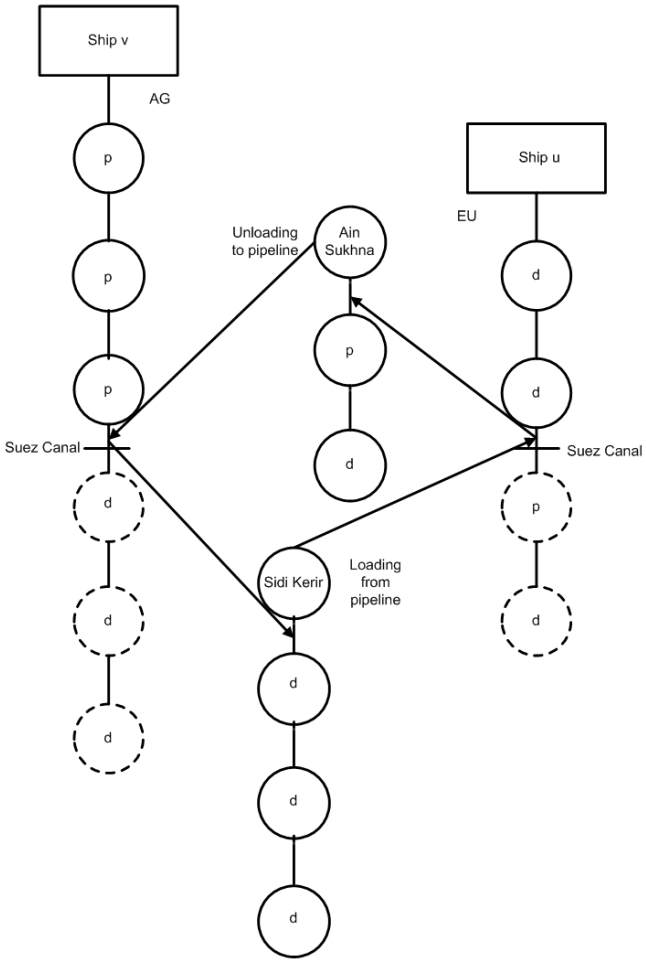

Figure 9.4: Suez-Sumed Heuristic

Figure 9.4, shows a possible voyage interchange heuristic. It principally works much in the same manner as the 2-interchange, only it interchanges the rest of pick-ups and deliveries in the sailing plan of each vessel (ship u and ship v) in the example. Ship v is on route from doing pick-ups (nodes marked with p are pick-up nodes) in the Arabian Gulf (AG), and Ship u is returning from deliveries (nodes marked with d are delivery nodes) in Europe. In stead of both vessels passing through the Suez Canal, their sailing plans as they would have been after passing the canal are interchanged. In addition the nodes

Ain Sukhna and Sidi Kerir are added. Ain Sukna as a delivery port and Sidi Kerir as a pick-up port.

Alternatively the interchange could only include the rest of the current voyage for ship v, and next voyage for ship u. Remembering that a voyage is defined as a sequence of port calls, starting with the port where the vessel loads its first cargo and ending where the vessel unloads its last cargo and becomes available again. This way of interchanging may possibly create problems with regards to time windows.

9.3 Concluding remarks

Concluding there is a range of new heuristics to be created and tested for problems with decoupled pick-ups and deliveries. Such problems have an extra degree of freedom that can be exploited when trying to optimize them. A new insertion heuristic matching delivery, pick-up and ship has been presented along with a new version of the 2-interchange neighbourhood, and these can be regarded as examples of ways of utilizing the freedom available in the problem.

If such neighbourhoods were to be proven effective, one could also consider always trying to create this degree of freedom in problems that are not necessarily decoupled of nature. In problems like the one discussed, different products are transported between different ports. The same product type may be available for pick-up at different ports, before being delivered at a given set of ports. If pick-up and delivery is matched before assigning it to a vessel, which is the standard way in which things are done in real life, and if it is not made possible to break up this match, any solution derived may already be sub-optimal. Industrial shipping problems with the same goods available at several ports for pick-up and then is to be delivered at other ports , have the characteristics necessary to possibly benefit from removing the constraint of having pick-ups and deliveries that are matched and merged.

The Suez Canal, is an interesting optimization obstacle. The problem including the Suez Canal and the Sumed pipeline has been described, and a possible heuristic has been presented in Figure 9.4. This Suez-Sumed heuristic reduces the use of the Suez canal, possibly resulting in reduced costs .

On a final note, and outside the scope of the report are strategic considerations and using a holistic approach when optimizing ship scheduling and routing problems. The fleet could be dimensioned to service the needs of the shipper. Pick-ups and deliveries should be planned and time-windows should be booked to fit the fleet at hand, with regards to size, sailing speed, dead weight and availability. If deliberate use of the Sumed Pipeline

is deemed beneficial, strategic measures like permanently positioning parts of the fleet on each side of the canal may be an interesting approach to the problem.

References

Brønmo, G., Christensen, M., Fagerholt, K. & Nygreen, B. (2007), 'A multi-start local search heuristic for ship scheduling - a computational study', *Computers & Operations Research* **34**, 900–917.

Bräysy, O., Gendreau, M., Hasle, G. & Løkketangen, A. (2003*a*), 'A survey of heuristics for the vehicle routing problem, part i: Basic problems and supply side extensions', *Technical Report, Molde University College* .

Bräysy, O., Gendreau, M., Hasle, G. & Løkketangen, A. (2003*b*), 'A survey of heuristics for the vehicle routing problem, part ii: Demand side extensions', *Technical Report. Molde University College* .

Christensen, M., Fagerholt, K., Nygreen, B. & Ronen, D. (2007), *Maritime Transportation, Handbook in operations research and management science: transportation*, Vol. 14, Barnhart C, Laporte G, editors.

Christensen, M., Fagerholt, K. & Ronen, D. (2004), 'Ship routing and scheduling: Status and perspectives', *Transportation Science* **38(1)**, 1–18.

Clarke, G. & Wright, J. W. (1964), 'Scheduling of vehicles from a central depot to a number of delivery points', *Operations Research* **12(4)**, 568–581.

Cordeau, J.-F., Gendreau, M. & Laporte, G. (1997), 'A tabu search heuristic for periodic and multi-depot vehicle routing problems', *Networks* **30(2)**, 105 – 119.

Cordeau, J.-F. & Laporte, G. (2003), 'A tabu search heuristic for the static multi-vehicle dial-a-ride problem', *Transportation Research Part B* **37**, 579–594.

Cormen, T. H., Leiserson, C. H., Rivest, R. L. & Stein, C. (2001), *Introduction to algorithms, second edition*, The MIT Press.

Dumas, Y., Desrosiers, J. & Soumis, F. (1991), 'The pickup and delivery problem with time windows', *European Journal of Operation Research* **54**, 7–22.

Glover, F. W. & Laguna, M. (1997), *Tabu search*, Springer.

Lübbecke, M. E. & Desrosiers, J. (2005), *Column Generation*, US: Springer.

Lin, S. (1965), 'Computer solutions of the traveling salesman problem', *Bell Systems Technical Journal* **44**, 2245–2269.

Martí, R. (2003), *Multi-Start Methods, In Glover FW, Koechemberger GA, editors. Handbook of metaheuristics*, Berlin: Springer.

Nanry, W. P. & Barnes, J. W. (2000), 'Solving the pickup and delivery problem with time windows using reactive tabu search', *Transportation Research Part B* **34**, 107–121.

Potvin, J.-Y. & Rousseau, J.-M. (1995), 'An exchange heuristic for routing problems with time windows', *The Journal of the Operational Research Society* **16(12)**, 1433–1446.

Ronen, D. (1993), 'Ship schedulig: The last decade', *European Journal of Operational Research* **71(3)**, 325–333.

Ropke, S. & Pisinger, D. (2006), 'An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows', *Transportation Science* **40(4)**, 455–472.

Savelsbergh, M. W. P. & Sol, M. (1995), 'The general pickup and delivery problem', *Transportation Science* **29**, 17–29.

Solomon, M. M. (1987), 'Algorithms for the vehicle routing and scheduling problems with time window constraints', *Operations Research* **35(2)**, 254–256.