Bachelor Thesis: Econometrics and Operations Research
Student number: 383549
Date: 03-07-2016

Incorporating random delays into an existing vehicle schedule: a method comparison

Author: Tjebbe Bodewes[1]
Supervisor: Judith Mulder[1]
Co-reader: Chiel van Oosterom[1]

Abstract
In this thesis we address the problem of adapting an existing schedule to random delays by deciding on how to allocate available buffer time to different legs of the route and what actions to take in case delay has been incurred. We build upon the work of Mulder and Dekker (2016), who formulated the problem of deciding on the actions as a linear program based on a Markov Decision Process and presented a mixed integer programming formulation for the full problem. They also proposed two exchange heuristics, which compute or estimate the cost reduction for each possible exchange of buffer between legs and make the best exchange. We propose several more methods and recommend three of them based on computational experiments. The best slow and steady method is the exchange heuristic that computes cost exactly. A similar heuristic that does not analyse every exchange but draws them randomly should be used for fast approximations of an optimum. An exchange heuristic that does not recompute cost at every iteration, but uses previously computed values, provides the best balance between solution quality and computation time. In addition, we show that the estimation method is inaccurate and should be treated with care.

[1] *Erasmus School of Economics, Erasmus University Rotterdam, the Netherlands*

Contents

Introduction

Scheduling vehicles effectively can lead to drastic cost reductions and service level improvements in logistical networks. Unnecessary detours can be avoided by determining in what order locations of interest should be visited, and by also providing a time window in which these locations should be visited one reduces waiting time or problems in transferring cargo from one mode of transport to another. These scheduling problems are often large and complex, making it difficult to include stochastic elements, like delays. In reality however, delays are an important part of logistical operations and neglecting them can greatly diminish the efficiency improvement gained by scheduling in the first place. The fact that delay is important to take into account when obtaining a vehicle schedule, but hard to incorporate into scheduling models, motivates this thesis. For this problem, we analyze how to incorporate delays into a given schedule, separating this from the initial scheduling. Firstly, given some initial schedule, one can allocate buffer time to the various legs of the route. Secondly, for a final schedule where buffer has been allocated, one can decide what action should be taken in case delay is incurred. With this twofold approach, a schedule can be adapted to delays.

This thesis is based on the work of Mulder and Dekker (2016). They address the problem of choosing recovery actions and allocating buffer time for liner ships, presenting a mixed integer programming formulation that guarantees optimality and several heuristics. In this thesis, we will also address the problem from the perspective of shipping when necessary. This is mainly because recovery actions between ports are relatively easy for ships as they are not encumbered by traffic or tracks. However, we aim to present the methods and results a general fashion, such that they can also be applied to other modes of transport.

First we will discuss papers on disruption management in the literature review, followed by a discussion of the problem at hand and the theoretical framework required for it in the problem description section. Here the MIP formulation by Mulder and Dekker (2016) is also presented, In the next section we derive some theoretical results on the problem. After this, we discuss several heuristics in the section on heuristics and compare their performance in the section on computational experiments. We also study several other aspects of the methods and the problem, like whether reliable estimates for the cost reduction due to exchanging buffer between ports can be made and what the shadow price of additional buffer time is. In addition, we discuss a maximum likelihood approach to estimating the mean computation time in case an algorithm is terminated prematurely in some cases, due to a time limit being exceeded. The aim of this thesis is to find out how high quality solutions of the vehicle schedule recovery problem with buffer allocation can be obtained efficiently. In particular, we aim to find out whether the methods provided by Mulder and Dekker are satisfactory and see if other methods are better. We also want to recommend a quick method that gives reasonable solutions, a balanced method that gives good solutions and a slow method that gives very good solutions.

1. Literature review

Disruption management is a topic that has already been well studied in some industries, in particular for aviation and rail transport. An example is Thengvall, Bard and Yu (2000) who discuss recovering delays for a fleet of aircraft by canceling flights or by letting delay propagate through the schedule until it has been fully absorbed. They present an MIP formulation in which user preferences, such as minimal deviation from the original schedule, can also be incorporated, and show that the LP relaxation combined with a rounding heuristic gives good solutions to the problem. One interesting thing that they mention, but do not consider further is to look at disruption management from the perspective of individual passengers and look at missed connections. Another example is Walker, Snowdon and Ryan (2005) who address disruption management on a rail network. They deal with the additional complication that changing the vehicle schedule also requires changing the crew schedule and solve these problems jointly. They give an integer programming formulation for constructing a new schedule based on the old one and show that it solves the problem in an acceptable amount of time. Naumann, Suhl and Kramkowski (2011) discuss a stochastic programming approach to robust vehicle scheduling in public bus transport, which outperforms both scheduling without buffer times and allocating fixed buffer times between service trips in terms of expected cost. They mention that because buffer times cause more 'planned' cost, there tends to be little buffer time in ordinary schedules and show that expected cost can be reduced significantly by incorporating buffer time in a clever way.

Schedule recovery for shipping has not been studied exhaustively yet. For an overview of research on liner ship routing and scheduling from the past 30 years, we refer to Meng, Wang and Thun (2012). Notteboom (2006) discusses four main categories of delays in shipping: terminal operations, port access, maritime passages and chance. On the Asia-Europe line studied in Notteboom's paper, terminal operations are by far the largest cause of delay, accounting for 85% of schedule unreliability. In the same paper, several recovery actions are discussed: port swapping, port skipping, deploying vessels to take over, decreasing the turnaround time in some ports, or increasing the sailing speed. A more technical treatment of the Vessel Schedule Recovery Problem (VSRP) is given in Brouer et al. (2013). In this problem a vessel incurs delay once and at this point a recovery action should be picked, such that recovery takes place within a fixed number of steps on the route. They discuss that container ships are in some ways similar to aircraft, where ships are like aircraft and containers are like passengers. As disruption management has been studied more extensively in aviation than in shipping, they propose to use similar methods as the aviation industry does. They also highlight some differences, such as that the recovery method of delay propagation used for airlines is not applicable for ships, while the shipping equivalents of vessel swapping and trip cancellation are port swapping and port cancellation. They model the problem on a directed graph on a time-space network and give an MIP formulation. They also show that the VSRP is NP-hard.

Disruption management for ships is also discussed in Qi (2015). He mentions the difference between delay like port congestion, which can be expected, and delay due to extreme events, like typhoons. The first type motivates incorporating buffers in the schedule, while the second type motivates the use of recovery actions. Recovering from delays by increasing speed and skipping or swapping ports is studied extensively by Li, Qi and Lee (2015). They derive structural results for the optimal schedule in case only speeding up is allowed and formulate this case as a non-linear program. They give a dynamic programming algorithm for cases where port skipping is allowed as well and propose a local search heuristic for when both port swapping and port skipping is allowed.

2. Problem description

Consider a single vehicle on a route consisting of N legs, where each leg consists of a location and the trip to the next location, but does not include this next location. The planned arrival and departure times are known for each location, but there is a positive probability of incurring delays while traveling between locations. The problem at hand is to allocate available buffer time in advance and to decide what recovery actions should be taken when delay has been incurred. In general, possible recovery actions are speeding up, changing the order in which locations are visited or skipping locations. We will only treat the possibility of speeding up, but the methods can easily be extended to include the other cases. The objective is to minimize the expected cost, consisting of the cost of delays and the fuel cost. First, some theory on Markov Decision Processes (MDP) is discussed, as it is the framework that we will apply to the problem at hand. Then we treat the problem of choosing recovery actions for a given buffer allocation. In the section thereafter we extend this to finding an optimal buffer allocation in addition to the optimal recovery actions.

2.1 Theory of Markov Decision Processes

At the core of the problem at hand is the random nature of the delays. To find a good schedule, we need a framework that allows for actions to be chosen in an environment that involves stochastic elements. A discrete Markov Decision Process, a generalization of the Markov process, is well suited for this task. We will now discuss the results on MDP's that we require, for a more extensive treatment we refer to Puterman (2005) and White and White (1989). For further reading on normal Markov processes we refer to Ross (2014).

A discrete Markov Decision process is a collection of objects $(\mathcal{T},\ \mathcal{I},\ \mathcal{A}_i,\ p(\cdot|i,a),\ r(i,a))$. Here $\mathcal{T} = \{1, 2, \dots, T\}$ is the set of decision epochs. For our purpose, we consider an infinite horizon: $\mathcal{T} = \{1, 2, \dots\}$. In each decision epoch the MDP is in a state $i \in \mathcal{I}$ and a decision $a \in \mathcal{A}_i$ has to be made. At this point the state changes to some other state $j \in \mathcal{I}$ with probability $p(j|i,a)$ and a reward $r(i,a)$ is obtained. We restrict ourselves to cases where $\mathcal{I}$ and all sets $\mathcal{A}_i$ are finite (and thus discrete). In MDP's, one requires a procedure to choose actions. For this, we use the idea of a decision rule, which corresponds an action to each state at each decision epoch. A deterministic decision rule takes the form of a function $d : \mathcal{I} \to \mathcal{A}_i$. A policy is a set of decision rules $(d_1, d_2, \dots)$, giving a decision rule for each decision epoch. We only consider stationary policies $\delta = (d, d, \dots) = \{d\}^\infty$ with the same decision rule in each epoch. Our central question is what policy should be used to achieve optimality, given some optimality criterion. Given some deterministic policy, let $\{X_t^\delta, t \in \mathcal{T}\}$ be the Markov process induced by that policy. Now the reward in decision epoch t is a random variable given by $r(X_t^\delta, d(X_t^\delta))$.

The optimality criterion that we use is the average reward criterion. In addition to the assumption that we have already made ($r(i,a)$ and $p(j|i,a)$ not time dependent, and $\mathcal{I}$ finite) we assume that $|r(i,a)| \leq M < \infty$ for all states i and actions a. Let $v_T^\delta(i)$, given by Equation 1, denote the total reward up to decision epoch T if the MDP starts in state i and policy δ is used.

$$v_T^\delta(i) = E_i\left[\sum_{t=1}^{T} r(X_t^\delta, d(X_t^\delta)\right] \tag{1}$$

Here E_i means expectation over the distribution of X_t^δ given that $X_1^\delta = i$. Let $g^\delta(i)$, displayed in Equation 2, be the average reward function given some policy δ and initial state i.

$$g^\delta(i) = \lim_{T\to\infty} \frac{1}{T} v_T^\delta(i) \tag{2}$$

As mentioned before, a stationary policy $\{d\}^\infty$ is followed. Let P_d denote the transition probability matrix induced by decision rule d. Its limiting matrix P_d^* exists if $\mathcal{I}$ is countable and is given by Equation 3.

$$P_d^* = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} P_d^{t-1} \tag{3}$$

A real square matrix is called (right) stochastic if all its entries are positive and each of its row sums equals one. If $\mathcal{I}$ is countable and P_d^* is stochastic, the limit in Equation 2 exists. That $\mathcal{I}$ is countable follows from that it is finite and we assume that P_d^* is stochastic for all decision rules d. Now we require a method that finds an optimal policy δ^* from the set Δ of stationary policies. An optimal policy is defined as in Equation 4. Note that this definition requires $g^\delta(i)$ to be constant over all initial states for all $\delta \in \Delta$. Puterman (2005) shows that this holds if P_d^* is stochastic for all decision rules d and the Markov chain implied by a policy δ consists of a single recurrent class and possibly some transient states for all $\delta \in \Delta$. We assume both of these and give a motivation for the latter in the next section.

$$\delta^* = \arg\sup_{\delta\in\Delta}\{g^\delta(i)\} \tag{4}$$

Various methods can be used to find δ^*, like value iteration, policy iteration or linear programming. We will follow the linear programming approach, as it is most easily related to the MIP formulation that we present later. Note however that value or policy iteration are often a more efficient way to find optimal policies. Puterman (2005) shows that the problem can be formulated as on the next page.

$$\max \sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i} r(i,a)x(i,a) \tag{5}$$

$$\text{s.t.} \sum_{a\in\mathcal{A}_j} x(j,a) - \sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i} p(j|i,a)x(i,a) = 0,\ j\in\mathcal{I} \tag{6}$$

$$\sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i} x(i,a) = 1 \tag{7}$$

In this formulation $x(i,a)$ can be interpreted as the steady state joint probability of being in state i and choosing action a, therefore $x(i,a)\geq 0$ for all states and actions. Now the objective function gives the average reward per decision epoch. The restrictions in Equation 6 function as balance equations, relating the steady state probabilities to each other, while Equation 7 ensures that the probabilities sum to 1. As mentioned, we assume that the MDP that we consider is unichain, meaning that there is a single class of recurrent states and a possibly empty set of transient states in every Markov chain induced by a deterministic, stationary policy. Puterman (2005) proves that a bounded, optimal basic feasible solution x^* exists and that it can be translated into a deterministic stationary policy that satisfies Equation 4. Define the set $\mathcal{I}_{x^*} = \{i\in\mathcal{I}:\ \sum_{a\in\mathcal{A}_i} x^*(i,a) > 0\}$. For all $i\in\mathcal{I}_{x^*}$ there exists exactly one $a\in\mathcal{A}_i$ such that $x(i,a)>0$. Now an optimal policy is given by $\delta^* = (d_{x^*}, d_{x^*}, \dots)$, where $d_{x^*}(i) = a$ if $x(i,a)>0$ and $i\in\mathcal{I}_{x^*}$ and arbitrary otherwise. In other words, we obtain an action for each state that is recurrent in the Markov chain induced by the optimal policy.

2.2 Recovery actions

For the problem of choosing recovery actions without allocating buffer, we define the following notation:

$\mathcal{L}$:= Legs of the route
$\mathcal{D}$:= Set of possible delays
$\mathcal{K}_l$:= Set of possible actions during leg l
$\mathcal{K}$:= Set of all possible actions, $\mathcal{K} = \cup_{l\in L}\mathcal{K}_l$

We assumed that the legs of the route are ordered in a circular manner: after finishing the last leg a vehicle starts the first again. To make $\mathcal{D}$ a finite set, we only consider discrete time units, instead of a continuum. Additionally, allowed delay should have an upper bound $\bar{d}$. The cost of having more delay than $\bar{d}$ is defined to be very high, such that it is always cheaper to choose recovery actions such that the probability of ending up with more delay than $\bar{d}$ is low. Therefore we need only a few delays in D which are larger than $\bar{d}$, such that $\mathcal{D} = \{d\in\mathbb{N}\mid d\leq \bar{d}+x\}$, where x is a small integer. One can model the problem as an MDP, with states $\mathcal{I} = \mathcal{L}$ x $\mathcal{D}$ consisting of both the leg and the current delay. In each state $i\in\mathcal{I}$, an action $k\in\mathcal{K}_i$ is chosen. Here $\mathcal{K}_i$ is the set of available actions in state i. In our model, it is not necessary to have different actions available at different delays, so if $i=(l,d)$, $\mathcal{K}_i = \mathcal{K}_l$. There is a cost associated with being in a state, the delay penalty, and a cost associated with choosing a certain speed in that state. Taking the sum of the delay cost and the fuel cost, we define C_{ik} as the total cost of choosing action k in state i.

Let p_{ijk} be the probability of transitioning from state i to state j if action k is chosen and let $\bar{p}_{iq}$ be the probability that q time units of additional delay are incurred when the Markov chain is in state $i=(p,d)$. Let g_k be the gain with respect to the schedule in time units of recovery action k and b_l be the number of buffer time units on leg l. The Markov chain moves from state $i=(l,d)$ to state $j=(l',d')$ if l' is the leg after l and $d' = d+q-g_k-b_p$, where q is the delay. The transition probabilities in case l' is the leg after l are defined as Equation 8 if $d'>0$ and as Equation 9 if $d'=0$.

$$p_{ijk} = \bar{p}_{iq} \text{ with } q = q = d' - \min\{d,\bar{d}\} + g_k + b_l \tag{8}$$

$$p_{ijk} = \sum_{q\in D}\bar{p}_{iq} \text{ with } D = \{q\mid d+q-g_k-b_l\leq 0\} \tag{9}$$

Define the set of decision epochs as $\mathcal{T} = \{1,2,\dots\}$. The MDP is now $(\mathcal{T},\mathcal{I},\mathcal{K}_i,p_{ijk},-C_{ik})$. We assume that all Markov processes generated from this MDP by a policy are unichain (single class of recurrent states and a possibly empty set of transient states). Solving for the steady state probabilities for a number of policies shows that this assumption seems to hold. We will not prove that it holds, but give a brief argument. First, you can reach any leg from any other leg with positive probability in a finite number of steps due to the circular nature of the route. As for the delay, in some cases you gain more time with respect to your schedule than the average delay, but you cannot have less than zero delay. In these cases the process will mostly be confined to the states with low delay and never reach certain states with high delay. On the other hand, if the expected delay is at least as much as the gain, the process will mostly be confined to states of high delay and never reach certain states with low delay. Because whether two states communicate only depends on whether one can move from the delay of one state to the delay of the other state and the process tends to drift to high or low delay, depending on the policy, it is unlikely that there are two separate classes of communicating states. This gives some justification for the unichain assumption. Let $\pi_{ik} = P(\text{state = i and action = k})$ be the long run steady state probabilities. One can now formulate the problem as a linear program:

$$\min |\mathcal{L}| \sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}_i} C_{ik}\pi_{ik} \tag{10}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}_i}\pi_{ik} = 1 \tag{11}$$

$$\sum_{k\in\mathcal{K}_j}\pi_{jk} - \sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}_i}\pi_{ik}p_{ijk} = 0 \qquad j\in\mathcal{I} \tag{12}$$

$$\pi_{ik}\geq 0 \qquad i\in\mathcal{I},\ k\in\mathcal{K}_i$$

The objective function in Equation 10 is the expected cost for the full route. The part with the double summation is the average cost per decision epoch and in traversing the full route, one has exactly $|\mathcal{L}|$ decision epochs. The meaning of the restrictions is the same as those in the general LP formulation for MDP's, presented in the previous section. The optimal solution obtained from this formulation can be translated to a deterministic decision rule in the manner described previously. As for the size of the problem, there are $\sum_{i\in\mathcal{I}}|\mathcal{K}_i|$ real variables and $|\mathcal{I}|+1$ constraints. When only considering speeding up, for state $i=(l,d)$ we have $\mathcal{K}_i=\{t\in\mathbb{N}: \frac{d_l}{a}\leq t\leq \frac{d_l}{b}\}$, with a and b the maximum and minimum speed respectively and d_l the distance to be traveled in leg l. Therefore $|\mathcal{K}_i|=\lfloor\frac{d_l}{b}\rfloor-\lceil\frac{d_l}{a}\rceil$, independent from the delay in state i. In case this would constitute an empty set (d_i too small), one action can be added, in which the vehicle travels at minimum speed. Ignoring this, the total number of variables can thus be written as $|\mathcal{D}|\sum_{l\in\mathcal{L}}\left(\lfloor\frac{d_l}{b}\rfloor-\lceil\frac{d_l}{a}\rceil\right)$.

2.3 Buffer time allocation

Intercontinental liner shipping routes are usually serviced once a week (see for example Notteboom (2006, p. 22)), but the minimum amount of time necessary to visit all ports is in general not an integer number of weeks. Therefore, the scheduled time is often rounded up to be an integer number of weeks, meaning that there is more time available than strictly necessary. This time is called buffer time and therefore besides choosing recovery actions when delay has been incurred, it is possible to allocate this buffer time to the legs. For other modes of transport, buffer time could be made available deliberately. As the allocating is done in advance, the buffer time allocation needs to be independent of the realized delay. This means that one cannot treat these buffer times in the same way as the recovery actions in an MDP framework. Mulder and Dekker (2016) presented a mixed integer programming formulation for this problem, based on the linear program presented earlier. Following their notation, I introduce the following additional sets:

$\mathcal{B}$:= Set of possible amounts of buffer times
$\mathcal{A}_i$:= Set of possible actions in state i of the new MDP
$\mathcal{A}$:= Set of possible actions in the new MDP

As an action will now consist of both a recovery action and a buffer time, it follows that $\mathcal{A}_i=\mathcal{K}_i\ x\ \mathcal{B}$, meaning that every action $a\in\mathcal{A}$ is a pair $a=(k,b)$. Let M be the total amount of buffer time available for the entire route and let B_b be the number of time units in buffer $b\in\mathcal{B}$. Finally, if $a=(k,b)$, let p_{ija} be defined in the same way as p_{ijk}, but with g_a instead of g_k. In this case, g_a+b_l denotes the time gain with respect to the original schedule without buffer allocation, due to both the recovery action k and the buffer time b. The problem at hand can be formulated as follows:

$$\min |\mathcal{L}|\sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i}C_{ia}\pi_{ia} \tag{13}$$

$$\text{s.t.}\ \sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i}\pi_{ia}=1 \tag{14}$$

$$\sum_{a\in\mathcal{A}_j}\pi_{ja}-\sum_{i\in\mathcal{I}}\sum_{a\in\mathcal{A}_i}\pi_{ia}p_{ija}=0 \qquad j\in\mathcal{I} \tag{15}$$

$$\sum_{l\in\mathcal{L}}\sum_{b\in\mathcal{B}}B_b y_{lb}\leq M \tag{16}$$

$$\sum_{b\in\mathcal{B}}y_{lb}=1 \qquad l\in\mathcal{L} \tag{17}$$

$$\sum_{d\in\mathcal{D}}\sum_{k\in\mathcal{K}_{(pd)}}\pi_{(ld),(kb)}=\frac{y_{lb}}{|\mathcal{L}|} \qquad l\in\mathcal{L},\ b\in\mathcal{B} \tag{18}$$

$$\pi_{ia}\geq 0 \qquad i\in\mathcal{I},\ a\in\mathcal{A}$$

$$y_{pb}\in\{0,1\} \qquad p\in\mathcal{P},\ b\in\mathcal{B}$$

The objective function and first two restrictions have been explained earlier. An additional binary decision variable y_{lb} is introduced, which is one if buffer b is allocated to leg l and zero otherwise. Equation 16 ensures that no more than the total available buffer time is allocated, while Equation 17 makes sure that only a unique buffer time is chosen for each leg. Equation 18 enforces that a certain buffer time 'action' b can only be taken on leg l if that amount of buffer time has been allocated to this leg. The equality in this restriction holds because the ship is in each leg with equal probability. Besides the variables and constraints that were also in the LP formulation, an additional $|\mathcal{L}||\mathcal{B}|$ integer variables and $1+|\mathcal{L}|+|\mathcal{L}||\mathcal{B}|$ constraints appear in this formulation.

3. Theoretical results

Number of possible buffer allocations

To motivate the extensive treatment of heuristics in the next section, I will now discuss an aspect of the computational complexity of the problem at hand. By considering the total number of buffer configurations, one sees that enumeration is not feasible. Each buffer allocation is a specific way of allocating M buffer units to $N = |\mathcal{L}|$ bins, where the order of allocation is not important in a combinatorial sense and where multiple buffer units can be allocated to the same bin. This is a clear case of a combination with repetition. The total number of possibilities is therefore given by Equation 19. This can also be seen in the following way: if z_l is the amount of buffer allocated to port l, then there exists a one-to-one relationship between the set of feasible buffer allocations and the set of non-negative integer solutions to the Diophantine equation $z_1 + \cdots + z_N = M$. It is known that the number of such solutions is given by Equation 19 and the number of feasible buffer allocations is the same due to the one-to-one relation.

$$\binom{M+N-1}{M} = \binom{M+N-1}{N-1} = \frac{(M+N-1)!}{M!(N-1)!} \tag{19}$$

By using Stirling numbers of the first kind s_{ij} (see for example Abramowitz and Stegun, 1972) we can expand a binomial coefficient around a fixed point n_0 as in Equation 20, with the coefficients of the polynomial given by Equation 21.

$$\binom{n}{k} = \sum_{i=0}^{k} c_i (n - n_0)^i \tag{20}$$

$$c_i = \sum_{j=i}^{k} n_0^{j-i} \binom{j}{i} \frac{s_{ij}}{k!} \tag{21}$$

This shows that a binomial coefficient grows polynomially in n if k is considered to be fixed. Given Equation 19, this means that the number of feasible buffer allocations is polynomial in N if M is fixed, or polynomial in M if N is fixed (due to the property of binomial coefficients that $\binom{n}{k} = \binom{n}{n-k}$).

However, polynomial growth in the numeric value does not mean polynomial complexity. A computational complexity of $\mathcal{O}(f(n))$ means that the number of computations grows (asymptotically) proportional to $f(n)$, with n the number of bits required to represent the input. If we want to represent a number x in binary, this takes $n = \mathcal{O}(log_2(x))$ bits. We can then write $x = \sum_{i=0}^{n} b_i 2^i$ for some set of coefficients $b_i \in \mathbb{B}$. This means that $x = \mathcal{O}(2^n)$. Therefore, even if the number of computations is polynomial in the input, $\mathcal{O}(x^k)$, it is exponential in the number of bits, $\mathcal{O}(2^{kn})$. Garey and Johnson (1978) noted this problem and named algorithms of which the computation time grows polynomially in the numeric value of the input 'pseudo-polynomial'. This means that if we consider either N or M to be a fixed value instead of part of the input of the problem, the number of MDP's that have to be solved grows pseudo-polynomially in the other parameter. It is known that MDP's can be solved in polynomial time (see for example Papadimitriou and Tsiksitlis, 1987). This means that enumeration would have pseudo-polynomial complexity if either the number of legs or the total amount of buffer is fixed. In Table 1 we display the order of magnitude of $\binom{M+N-1}{M}$ for some values of M and N. We can see that for a fixed N or M, the number does not seem to grow very rapidly in the other parameter. On the other hand, considering that an MDP would have to be solved for each possible allocation, enumeration is clearly not feasible in practice.

N\M	5	10	15	20
5	1E+02	1E+03	4E+03	1E+04
10	2E+03	9E+04	1E+06	1E+07
15	1E+04	2E+06	8E+07	1E+09
20	4E+04	2E+07	2E+09	7E+10

Table 1. Number of M-combinations from N with repetition

Cost of actions

In determining the cost of specific actions for computational experiments, we will assume that a vehicle moves at constant velocity. It seems intuitively correct that this gives the lowest fuel cost, but we will now show that it is indeed reasonable to assume. This requires some other assumptions on the fuel cost per hour of the vehicle. The first is that the cost depends only on the speed and not on position or time, the second is that the cost is increasing in the speed and the third is that the cost is convex in the speed. We want to choose speed as a function of time such that total cost for a leg is minimized. Mathematically, let the position of the vehicle during a specific leg as a function of time be denoted by $x(t)$ in a reference frame such that $x(0) = 0$. Choosing an action such that T time units are allocated to this leg is equivalent to stating that $x(T) = d$, where d is the traveling distance to the next location. We take velocity as the time derivative of position $\dot{x}(t)$, using Newton dot notation, and assume $\dot{x}(t) > 0 \; \forall t \in [0, T]$. From now on, the argument t in $\dot{x}(t)$ is dropped for the sake of brevity. Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ map velocity to fuel cost per hour, where it holds that $f'(\dot{x}) > 0$ and $f''(\dot{x}) > 0$ for $\dot{x} > 0$. Taking the functional $J[x]$ as the total cost of sailing the leg of interest, we can formulate the problem as follows:

$$\begin{aligned} \min \quad & J[x] = \int_0^T f(\dot{x}) dt \\ \text{s.t.} \quad & x(0) = 0, \quad x(T) = d \end{aligned} \tag{22}$$

We apply the Euler-Lagrange equation (see for example Brinkhuis & Tikhomirov, 2011, p. 478) given by Equation 23. Any stationary point of Equation 22 should satisfy this equation for all $t \in [0, T]$. The first term is zero, as f does not explicitly depend on position. As f also does not depend on time explicitly, the partial derivative in the second term reduces to an ordinary derivative, displayed in Equation 24.

Applying the chain rule, we obtain that any stationary point of this problem must satisfy Equation 25.

$$\frac{\partial f}{\partial x} - \frac{d}{dt}\frac{\partial f}{\partial \dot{x}} = 0 \tag{23}$$

$$\Rightarrow \frac{d}{dt} f'(\dot{x}) = 0 \tag{24}$$

$$\Rightarrow f''(\dot{x}(t))\ddot{x}(t) = 0 \qquad \forall t \in [0, T] \tag{25}$$

As we assumed $f''(\dot{x}) > 0$, this equation is only satisfied for all $t \in [0, T]$ if $\ddot{x}(t) = 0$, implying that $\dot{x}(t) = a$ and $x(t) = at + b$. Applying the boundary conditions, we find that $x(t) = \frac{d}{T}t$. Therefore, traveling at a constant speed of $\frac{d}{T}$ is the only stationary point of the optimization problem given above. We cannot conclude that it is a global minimum, as the Euler-Lagrange equation is only a necessary condition. Proving that this solution is a global minimum is beyond the scope of this thesis, but a practical argument can be made that sailing at a constant speed $\frac{d}{T}$ is indeed optimal for a fuel consumption that is increasing and convex in speed. Total displacement is the area under the velocity curve, which has to remain d. Therefore setting speed lower than $\frac{d}{T}$ during some time requires it to be greater during some other time. Because $f'(\dot{x}) > 0$ the former leads to lower cost, while the latter leads to higher cost. However, because $f''(\dot{x}) > 0$ the cost decrease is smaller than the cost increase, meaning that deviating from constant speed should increase overall cost.

4. Heuristics

As shown in the previous section, finding recovery actions given a buffer allocation can be solved efficiently as a linear programming problem. However, if we also consider the problem of allocating buffer we obtain a mixed integer programming problem. It is known that integer programming decision problems are NP-complete and therefore integer optimization in general is NP-hard (see for example Garey and Johnson (1979)). It seems unlikely that an efficient method that guarantees optimality exists for this problem. Therefore we will present a number of heuristic methods for approximating an optimal buffer allocation. First I will discuss two straightforward local search algorithms, that iteratively improve some initial buffer allocation. Then I will go into some possible adaptations of these heuristics, to either decrease their computational complexity or to increase the probability of obtaining a global optimum.

4.1 Simple local search

Local search algorithms are heuristic methods for finding solutions minimizing some criterion in a finite solution space. They improve upon a current solution by searching for better solutions in its neighborhood. For an extensive treatment of the use of local search heuristics in combinatorial optimization, we refer to Lenstra (1997) These methods can be used on a wide variety of problems, as they require few assumptions and are easy to implement. On the other hand, because they only search the neighborhood of a solution, they are sensitive to getting stuck in a local minimum and also require the 'neighborhood' of a solution to be defined in the first place. An additional issue is mentioned by Johnson, Papadimitriou and Yannakakis (1988), who point out that there are classes of problems for which even local optimality of solutions cannot be determined in polynomial time. This can also occur if the neighborhood of a solution is inconveniently defined. When we propose some heuristics that improve upon simple local search, we will try to circumvent these issues.

For the buffer allocation problem we consider a b-exchange local search heuristic, where two solutions are neighbors if and only if one can be obtained from the other by exchanging b units of buffer time between two legs. Initial solutions are randomly generated from a uniform distribution over all possible buffer allocations, using algorithm 1. We will now discuss the Buffer Exchange Heuristic (BEH) and the Value Iteration Heuristic (VIH) as proposed by Mulder and Dekker (2016).

Initialize empty allocation;
for i = *1 to* M **do**
  Generate X as discrete U(1,N) random variable;
  Increment buffer in leg X by 1;
end
Return allocation;
Algorithm 1: Generating random buffer allocations

Buffer Exchange Heuristic

The BEH is a steepest-descent method, that considers all neighbors b-exchange neighbors of the current solution and moves towards the neighbor that gives the biggest cost decrease, if one exists. If no neighbor with lower cost than the current solution exists, b is decreased or, if b is already 1, the algorithm terminates. The cost is computed by solving an MDP for a given buffer allocation. In algorithm 2 we display a pseudo-code for the BEH, where b_{max} denotes the maximum buffer in a single port (possibly different from the total amount of buffer M). We take a 2-permutation of ports because we want to exchange buffer between different ports (no repetition) and exchanging from port x to port y is different from exchanging from port y to port x (order is important). In case a b-exchange is not possible because the port from which buffer is removed has less than b units, the cost should be $+\infty$. In each iteration $N(N-1)+1$ MDP's have to be solved, increasing quadratically in the number of legs (implying a pseudo-polynomial running time). The size of the MDP's also grows as the number of legs increases (remember that the number of variables contained a summation over legs and the number of constraints is linear in the number of states, which is linear in the number of legs). Therefore, one can expect computation time to be heavily influenced by the number of legs.

Data: Feasible buffer allocation;
Set b to 2^k with $k = \lfloor \log_2(b_{max}) \rfloor$;
Set current allocation to given allocation;
while $b \geq 1$ **do**
  Compute cost of current allocation;
  Set best cost to current cost;
  Set best allocation to null;
  for *each 2-permutation of ports* **do**
    Exchange b buffer units between the two ports;
    Compute cost of new allocation;
    if *new cost* $<$ *best cost* **then**
      Set best cost = new cost;
      Set best allocation to the new one;
    end
  end
  if *best allocation* $\neq$ *null* **then**
    Set current allocation to best allocation;
  else
    $b = \frac{b}{2}$;
  end
end
Return current allocation;

Algorithm 2: Buffer Exchange Heuristic

Value Iteration Heuristic

The most problematic part of the BEH is that in each iteration a large number of MDP's have to be solved. Mulder and Dekker proposed a way of estimating the cost after changing an allocation slightly, using the value function. The value function is a mapping $v : \mathcal{I} \to \mathbb{R}$, mapping each state to the expected cost if one were to start in that state and follow an optimal policy from that point onward, up until the last leg of the route. This cost consists of the cost of being in the current state, the cost of taking the optimal action in this state and the expected cost for the rest of the tour. Therefore the value function is as given in Equation 26 for all states up until the states corresponding to the last leg. For states corresponding to the last leg, the value function is just $v_i = C_{ik^*(i)}$, where $k^*(i)$ denotes the optimal action in state i. Even though the value function originates from dynamic programming, backward induction cannot be used in a straightforward manner to determine the optimal recovery actions. This is due to the circular nature of the route: in the states corresponding to the last leg one requires the value function to be known for the states corresponding to the first leg. For our purpose, we determine the optimal actions by solving the MDP using linear programming and translate the solution to a value function. Alternatives would be more sophisticated implementations of the dynamic programming concept, like value- or policy iteration (see Puterman, 2005).

$$v_i = C_{ik^*(i)} + \sum_{j \in \mathcal{I}} p_{ijk^*(i)} v_j \tag{26}$$

This value function can be combined with the following insight: if an additional unit of buffer is allocated to a leg, two things may happen. Either the time allocated to that leg is increased by one, decreasing cost in the current leg, or the conditional distribution of the delay in the next leg shifts towards zero by one, decreasing cost in the future. Let $k'(i) = k^*(i) + 1$, where $k^*((l,d))$ is the optimal amount of time allocated to leg l with the current buffer allocation, given that one has delay d when starting this leg. We set $C_{ik} = \infty$ for all $k'(i) \notin \mathcal{K}(i)$ and $v_{l,-1} = v_{l,0}$ for all $l \in \mathcal{L}$. If $j = (l,d)$, define $j-1 = (l,d-1)$. If the value function for the current allocation is known, v_i^+ is an estimate of the value function for state i after adding one extra unit of buffer in the leg associated with that state. It is given by Equation 27. By taking $s_i^+ = v_i - v_i^+$ in Equation 28 we get the corresponding estimated cost reduction, which requires less operations to compute as some terms cancel out. This is also due to that the transition probabilities are not updated to match $k'(i)$ In these equations, let k' and k^* be understood to mean $k'(i)$ and $k^*(i)$ for ease of notation.

$$v_i^+ = \min\{C_{ik'} + \sum_{j \in \mathcal{I}} p_{ijk^*} v_j,\ C_{ik^*} + \sum_{j \in \mathcal{I}} p_{ijk^*} v_{j-1}\} \tag{27}$$

$$s_i^+ = \max\{C_{ik^*} - C_{ik'},\ \sum_{j \in \mathcal{I}} p_{ijk^*}(v_j - v_{j-1})\} \tag{28}$$

Now set $k'(i) = k^*(i) - 1$ and $v_{p,d^{max}+1} = v_{p,d^{max}} + C_p$, where C_p is the penalty cost per unit delay exceeding the maximum buffer. The estimated (negative) cost reduction s_i^- of allocating one less unit of buffer to the leg corresponding to i is given by Equation 29

$$s_i^- = \max\{C_{ik^*} - C_{ik'},\ \sum_{j \in \mathcal{I}} p_{ijk^*}(v_j - v_{j+1})\} \tag{29}$$

Up to now we only spoke about cost reduction per state, but buffer allocation is done per leg. To compute the cost reduction per leg, the distribution over states is required. More precisely, if S_l^+ is the cost reduction of adding one unit of buffer to leg l, it is given as in Equation 30. S_l^- is defined in a similar manner.

$$S_l^+ = \sum_{i \in \mathcal{I}} P(\text{state} = i \mid \text{leg} = l) s_i^+ \tag{30}$$

Denote the conditional probability in this equation by $p(i|l)$. In computing the value function of the current allocation an MDP has already been solved using linear programming. The required probabilities can be retrieved from its solution. Remember that $\pi_{ik} = P(\text{state} = i \text{ and action} = k)$ and that $P(\text{leg} = l) = \frac{1}{|\mathcal{L}|}$. Therefore $p(i|l)$ is given as in equation Equation 31.

$$p(i|l) = \begin{cases} |\mathcal{L}| \sum_{k \in \mathcal{K}(i)} \pi_{ik} & \text{if } \exists d \in \mathcal{D} : i = (l,d) \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

Using these probabilities, S_l^+ and S_l^- can be computed. The net cost reduction achieved by transferring one unit of buffer from leg x to leg y can therefore be estimated by $S_y^+ + S_x^-$.

The cost reduction is not computed exactly because neither the transition probabilities nor the distribution over states is updated. On the other hand, with this method the cost reduction of exchanging buffer can be estimated without solving an MDP, as long as the value function for the current allocation is known. This means that instead of solving $N(N-1)+1$ MDP's per iteration, only one needs to be solved (to determine the value function and optimal actions). Therefore a buffer exchange algorithm that uses the value function can be expected to work significantly faster than the BEH. Mulder and Dekker called this method the Value Iteration Heuristic (VIH) and it is presented in algorithm algorithm 3. Besides the cost estimation, there are two more differences with the BEH. The first is that in the VIH only 1-exchanges are considered, as otherwise the costs cannot be estimated accurately. The second is that the VIH uses a different stopping criterion. Instead of only making the best exchange if it is profitable, this exchange is even made if it leads to a cost increase. The solutions are stored and if the algorithm cycles and comes back to a solution that has already been considered, the algorithm terminates. In the algorithm we use 'saving' instead of 'cost reduction' for the sake of brevity.

Data: Feasible buffer allocation
Initialize empty list of solutions and their costs;
Set current allocation to the given one;
while *current allocation* $\notin$ *solutions* **do**
  Add current allocation and its cost to solutions;
  Solve MDP for current allocation;
  Compute value function for current allocation;
  Compute delay distribution for current allocation;
  Set best allocation to null;
  Set best saving to $-\infty$;
  for *each 2-permutation of ports* **do**
    Compute saving of exchanging 1 buffer unit;
    if *new saving* $>$ *best saving* **then**
      Set best saving = new saving;
      Set best allocation to the new one;
    end
  end
  Set current allocation to best allocation;
end
Return solution with lowest cost;

Algorithm 3: Value Iteration Heuristic

4.2 Adapted local search

Both of the methods discussed above were shown by Mulder and Dekker to give good solutions in far less time than the MIP formulation. However, neither are without flaw: the BEH is not very fast, while the VIH potentially leads to solutions that are far from the global optimum. Because both methods use local search, they are sensitive to local minima. I will now present several ways to improve upon the simple local search methods. For the BEH this will mainly be about decreasing the computation time, while for the VIH the focus lies on improving the quality of the solutions without compromising the running time too much.

Adapted buffer exchange

The main problem with the BEH is that in each iteration a large number of MDP's have to be solved. On this matter one might wonder whether it is necessary to solve a number of MDP's that is quadratic in the number of legs after each exchange. The idea behind the adapted buffer exchange heuristic (ABEH) is that for each value of b the saving is computed and stored once for each port combination. After this, only those that were found to give a cost reduction are considered in further iterations. In the first iteration, the exchange that gave the largest cost reduction is made. In the second iteration, the exchange with the second largest cost reduction is considered and an MDP is solved to see whether this would still lead to a cost reduction. If it does, the exchange is made, otherwise the next best exchange is considered. This goes on until there are no more positive savings in the list, at which point b is set to $\frac{b}{2}$. This means that $\mathcal{O}(N^2)$ MDP's have to be solved for each value of b. Therefore the total number of MDP's to be solved is in $\mathcal{O}(kN^2)$, where $k = \lfloor \log_2(b_{max}) \rfloor$. Compare this to the normal BEH, which has complexity $\mathcal{O}(N^2)$ for each iteration, while the number of iterations could be very large. As MDP's can be solved in polynomial time, the ABEH has pseudo-polynomial running time.

Stochastic buffer exchange

Even though the ABEH has to solve less MDP's, it still has to solve $\mathcal{O}(2kN^2)$. This is computationally intensive and not strictly necessary. Instead of trying all combinations of legs and selecting the best, one could also try random exchanges until one is found that gives a cost reduction. The advantage is also that instead of taking b to be decreasing with time, it can also be generated randomly, reducing the probability of getting stuck in a local minimum. This motivates the stochastic buffer exchange heuristic (SBEH). Given some initial allocation, it randomly generates a 2-permutation from the legs and a b and solves an MDP to find if exchanging b from one port to the other leads to a cost reduction. If it does, this exchange is accepted, the current allocation is updated and the process repeats. Each 2-permutation from the legs is equally likely to be drawn and the b is drawn uniformly from zero to the amount of buffer in the leg from which it will be removed, to guarantee that all ports have a non-negative amount of buffer.

The algorithm terminates if no improvement has been found for a certain number of iterations. An analysis of the complexity is less straightforward here, as the algorithm is probabilistic. An observation that can be made is that the running time per iteration of this algorithm does not (directly) depend on the number of legs, as opposed to all previously discussed algorithms.

Simulated annealing

The SBEH described above works quickly, but due to its hill-climber nature it might be prone to getting stuck in a local minimum. To reduce the probability of this happening, one could also accept worse solutions with a positive probability. If this is done in a particular way, it is called simulated annealing, as first described by Kirkpatrick and Vecchi (1983) and explained extensively as a method for operations research by Eglese (1990). Simulated annealing is based on physical annealing, slowly cooling molten material such that it gets to settle in a state of low energy (like a crystal). If the material is cooled too fast, the molecules do not get enough time to rearrange into a crystalline structure. This is because there are many 'local minima' for the molecules, that require energy to get out of. This energy is only available if the temperature is sufficiently high. For our purpose, this means that we start of with a high probability of accepting a worse solution (high temperature), but that this slowly decreases according to an annealing schedule, until at a very low temperature the method becomes equivalent to the stochastic buffer exchange heuristics.

In each iteration, a single possible exchange is drawn, the cost reduction is evaluated by solving an MDP and the exchange is accepted or rejected. Besides the aspects already described in the section on SBEH, one requires the acceptance probabilities and the temperature function (annealing schedule) to be specified. Let δ denote the change in cost due to the proposed exchange. If $\delta < 0$, the exchange reduces cost and is always accepted. If $\delta > 0$, the exchange is accepted with probability $e^{-\frac{\delta}{T}}$. This distribution is mentioned by many authors, including Eglese, because of its simplicity and its interpretation in thermodynamics and statistical mechanics. It is very similar to the Boltzmann factor $e^{-\frac{E_1-E_2}{qT}}$, which is the non-normalized probability that a physical system is in a state with energy level E_1 instead of one with energy level E_2, where q is the Boltzmann constant. We choose a simple annealing schedule, where the temperature in the kth iteration is given by $T_k = \alpha^k T_0$, with T_0 the initial temperature. The algorithm terminates when T_k falls below some predefined threshold and returns the best solution found in all iterations. This implies that the algorithm always does $k = \lceil \log_\alpha(T_0^{-1}) \rceil = \left\lceil \frac{\log_2(T_0)}{\log_2(\alpha^{-1})} \right\rceil$ iterations. Considering α to be fixed, the number of iterations is in $\mathcal{O}(n)$, where n is the number of bits required to represent T_0. As 1 MDP is solved in each iteration this algorithm has polynomial running time in the number of bits required to represent T_0.

Multi-start value iteration

A commonly applied method to reduce the probability that a local search method gets stuck in a local minimum is to run the algorithm for more than one starting point, after which the best solution out of all solutions is chosen. One can try to improve the quality of the solutions found by the VIH. The complexity grows linearly in the number of starting points, as for each starting point an optimization using the VIH has to be done.

5. Computational experiments

I will now test the methods described above using a combination of real data and randomly generated cases. The real data is largely based on the data described by Mulder and Dekker, on the ME1 route in September 2012 of the Maersk Line network.

5.1 Data

Port	Distance (nmi)	Sailing time (hr)
Jebel Ali	1329	60
Jawaharlal	443	20
Mundra	1122	52
Salalah	1553	68
Jeddah	778	36
Suez Canal	2283	100
Algeciras	1476	68
Felixstowe	156	8
Antwerp	366	16
Bremerhaven	283	16
Rotterdam	3829	168
Suez Canal	395	20
Aqaba	656	32
Jeddah	2648	116

Table 2. Route characteristics

The route that we use is given in Table 2. The first column is the name of the port in a leg (from which the ship departs), while the second column contains the distance in nautical miles (1 nmi $\approx$ 1.85 km) obtained from SeaRates (2015). The sailing time given in the third column is under the assumption that the ship sails at 23 knots (nmi/hr) and is rounded to a multiple of four. This is because we discretize time to units of four hours. Because liner shipping times are usually in the order of days or weeks, four hour blocks seem reasonable. We take the sailing time in the third column to be the 'default' schedule. Recovery actions and buffer allocations are made with respect to this schedule. Delay is capped at 20 time units (80 hours) and a total of 28 time units (112 hours) can be allocated as buffer time. For the ships, we assume a minimum and maximum speed of 12 and 23 knots respectively. For fuel cost in US dollar per day, we use the formula given by Brouer et al. (2014), displayed in Equation 32.

$$C_f(v) = \bar{f} * p_{bunker}(\frac{v}{\bar{v}})^3 \tag{32}$$

In this formula, $\bar{f}$ denotes the fuel consumption of the ship in ton per day when sailing at design speed, $\bar{v}$ the design speed of the ship in knots and p_{bunker} the bunker price in US dollars per ton. From Brouer et al, we obtain values for these parameters, using $p_{bunker} = \$600$ per ton, $\bar{v} = 16.5$ and $\bar{f} = 82.2$. Using these, the cost for a leg can be determined if the time allocated to it is known and ships are assumed to sail at constant speed. The cost $C_f^i(t_i)$ in US dollar of sailing the distance d_i of leg i in t_i hours is given in Equation 33. Note that this requires translating the blocks of time back into actual hours.

$$C_f^i(t_i) = \bar{f} * p_{bunker}(\frac{d_i}{t_i * \bar{v}})^3 * \frac{t_i}{24} \tag{33}$$

The cost per time unit of delay is taken to be \$10,000, implying a cost of \$2,500 per hour. The cost of each time unit of delay that exceeds the maximum delay is taken to be \$100 million. We take the default delay on leg i to follow a discrete uniform distribution between 0 and $2 + \lfloor \frac{d_i}{800} \rfloor$.

5.2 Schedule optimization

First we find an optimal schedule for the route given above, using the MIP formulation. The optimized costs are just over \$3.93 million and the optimized schedule is found by the CPLEX MIP solver in 1414 seconds. The schedule is given in Table 3.

Leg	Buffer (hr)	Delay (hr)			Min sailing time (hr)
		0	4	≥8	
Jebel Ali	12	68	64	60	60
Jawaharlal	8	24	20	20	20
Mundra	4	52	52	52	52
Salalah	12	76	72	68	68
Jeddah	4	36	36	36	36
Suez Canal	8	104	100	100	100
Algeciras	12	76	72	68	68
Felixstowe	4	8	8	8	8
Antwerp	8	20	16	16	16
Bremerhaven	4	16	16	16	16
Rotterdam	12	172	168	168	168
Suez Canal	8	24	20	20	20
Aqaba	4	32	32	32	32
Jeddah	12	124	120	116	116

Table 3. Buffer and sailing times for various delays under the optimized schedule

5.3 Method comparison

To compare the performance of our methods, we generated 70 test cases. Each of these had the same route as discussed before, but for each leg a discrete random $U(-1, 1)$ disturbance on the upper bound of the delay distribution was generated. Each case was solved using the MIP formulation, the BEH, the VIH with 1, 5 and 10 initial points, the ABEH, simulated annealing (SA) and the SBEH. For each test case, the computation time and the objective value of the obtained solution were recorded for all methods. It is important to note that the solution time also included any time required to build the model. All algorithms were capped at 3600 seconds, meaning that the computation times are right censored. Only the MIP formulation actually exceeded the 3600 second limit, meaning that it was terminated before finding an optimal solution. The costs are those required to complete a single full tour. Besides this 3600 second time limit, a number of other parameters had to be chosen, which we will discuss now.

Parameter settings

First of all, the optimality criterion for the MIP solver was that the gap between the best lower bound and the best upper bound was at most 0.1%. We also used the fact that random integer feasible solutions to the problem could be easily generated by providing the solver with 50 random initial allocations to start a branch and cut procedure from. The BEH and VIH do not require any parameters to be set, but the initial solutions were generated as mentioned before, with each possible allocation being equally likely. The ABEH does not require any parameters either, but an important detail is that all exchanges that were found to give a cost reduction at the start are analyzed. It might be worthwhile to consider other stopping criteria, like a certain number of iterations without improvement or considering all exchanges again (but ordered from largest to smallest cost reduction as found at the start). For simulated annealing, trial and error led to a parameter setting of $T_0 = 10000$ and $\alpha = 0,99$. The algorithm terminated if the temperature fell below 1. Finally, for SBEH the termination criterion was if 30 candidate exchanges in a row did not lead to a cost reduction.

Absolute performance

We will now look at the absolute performance over all cases of the methods separately. In Table 4 we report some of the key statistics on the methods. In the first row we display the average time required to either come to a solution or to terminate if the timer hits 3600 seconds. In the second row the coefficient of variation of this same measure is reported (the coefficient of variation is defined as $\frac{S}{\bar{X}}$, where S is the sample standard deviation and $\bar{X}$ is the sample mean). The latter is used as a measure of predictability of the computation time. We can see that in general the VIH-1, VIH-5 and SBEH methods tend to be the fastest, while the MIP and BEH are usually slower. Simulated annealing and the ABEH have quite predictable running times. For simulated annealing this can be explained naturally by that the number of iterations depends on T_0 and α, but not on the problem instance itself. For the ABEH it is likely due to that most of the running time can be attributed to evaluating the cost reduction of each possible exchange. This part has to be done the same number of times for each case.

	MIP	BEH	VIH-1	VIH-5	VIH-10	ABEH	SA	SBEH
Mean computation time (s)	1500	378	11	49	95	101	132	36
Coefficient of variation computation time (% of mean)	95%	26%	46%	32%	24%	13%	12%	38%
Max computation time (s)	3600	611	28	100	154	129	170	71
Proven optimal (% of cases)	74%	73%	0%	0%	0%	33%	16%	0%
Mean difference with MIP (% of MIP cost)		0.0%	1.2%	1.1%	0.4%	0.2%	0.1%	0.4%
Max difference with MIP (% of MIP cost)		0.2%	25.3%	23.3%	4.7%	9.7%	1.1%	7.7%
Fastest (% of cases)	0%	0%	100%	0%	0%	0%	0%	0%

Table 4. Key statistics on method comparison across 70 random cases

	MIP	BEH	VIH-1	VIH-5	VIH-10	ABEH	SA	SBEH
MIP		27%	0%	0%	0%	1%	6%	0%
BEH	73%		0%	0%	0%	0%	0%	0%
VIH-1	100%	100%		100%	100%	100%	100%	100%
VIH-5	100%	100%	0%		100%	100%	100%	21%
VIH-10	100%	100%	0%	0%		54%	99%	1%
ABEH	99%	100%	0%	0%	46%		99%	0%
SA	94%	100%	0%	0%	1%	1%		0%
SBEH	100%	100%	0%	79%	99%	100%	100%	

Table 5. Relative performance in computation time. (i,j) is the proportion of cases in which method i was faster than j

The computation times for the fastest three methods tend to be the most variable, but not nearly as unpredictable as the time required by the MIP solver. This is a problematic feature of the MIP method, which becomes even more clear when one considers Figure 1.

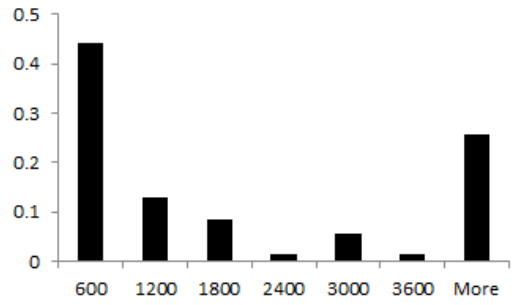

Figure 1. Frequency diagram for MIP computation times

The branch and cut process finds a solution within 10 minutes in 45% of the cases, which could be considered reasonable. However, in 30% of the cases, the algorithm did not find an optimal solution within an hour. This shows the flaw of MIP for this problem: the computation time is very unstable. This can be explained by the nature of the branch and cut method that is used, where the entire solution space has to be explored. By branching efficiently and pruning branches in which the optimal solution can not possibly be found, an attempt is made to render this possible. However, if the wrong variable is branched on somewhere at the start or large branches cannot be pruned, the running time increases very rapidly. This is another practical motivation for the use of heuristics. Based on this table, we can put these algorithms into three categories. The fast and sloppy algorithms are VIH-1, VIH-5 and SBEH, which tend to terminate in less than 100 seconds, but can give poor quality solutions. The balanced algorithms are VIH-10, ABEH and SA, giving good solutions in less than 200 seconds. Finally, the slow but steady algorithms are MIP and BEH, which run for a while, but usually give optimal solutions within an hour.

In the fourth row we show the percentage of cases in which the methods find an optimal solution. This optimality is determined as follows: if the MIP terminates within an hour, this solution is considered optimal. If another method obtains the same solution, this is also considered optimal. MIP finds the optimal solution within an hour in 52 out of 70 cases and BEH has the same solution in 51 of these 52 cases. This is in line with the finding of Mulder and Dekker that the BEH usually converges to an optimal solution. The ABEH also performs reasonably well, finding an optimal solution in 23 out of 70 cases. The next two rows show the difference between the MIP solution and the heuristic solution. Once again, the BEH finds the best solutions on overall, being at most 0.2% higher than the MIP solution. An important flaw of VIH can be found here: it can be off by a significant amount. Starting with a larger number of initial points seems to work well against this problem, reducing the worst case difference to 4.7%. The ABEH does well on average, but can be off quite far in the worst case scenario (but not nearly as far as VIH). Here the power of SA becomes apparent, as it clearly is less likely to get a bad solution than the SBEH and ABEH on which it is based. VIH-1 is always the fastest, as could be expected.

	MIP	BEH	VIH-1	VIH-5	VIH-10	ABEH	SA	SBEH
MIP		10%	100%	100%	100%	61%	81%	100%
BEH	7%		100%	100%	100%	66%	81%	97%
VIH-1	0%	0%		23%	14%	1%	0%	16%
VIH-5	0%	0%	71%		24%	3%	1%	27%
VIH-10	0%	0%	84%	69%		3%	10%	39%
ABEH	1%	0%	99%	97%	96%		59%	89%
SA	0%	1%	99%	96%	89%	24%		81%
SBEH	0%	0%	83%	71%	57%	7%	19%	

Table 6. Relative performance in solution quality. (i,j) is the proportion of cases in which method i gave a lower cost than j

	MIP	BEH	VIH-1	VIH-5	VIH-10	ABEH	SA	SBEH
MIP		1%	0%	0%	0%	1%	3%	0%
BEH	7%		0%	0%	0%	0%	0%	0%
VIH-1	0%	0%		23%	14%	1%	0%	16%
VIH-5	0%	0%	0%		24%	3%	1%	7%
VIH-10	0%	0%	0%	0%		3%	10%	0%
ABEH	1%	0%	0%	0%	46%		59%	0%
SA	0%	1%	0%	0%	1%	1%		0%
SBEH	0%	0%	0%	57%	56%	7%	19%	

Table 7. Overall relative performance. (i,j) is the proportion of cases in which method i gave a better solution in less time than j

Relative performance

We will now decide on a best algorithm for each of the three categories mentioned previously. To do this it is not sufficient to look at measures that are averaged over all cases. In this section we will analyze how the methods perform compared to each other on specific cases. For this purpose we use Table 5, Table 6 and Table 7. In these tables, the element at row i and column j is the proportion of cases in which the method at i was faster than the method of j, got a better solution or did both, for Table 5, Table 6 and Table 7 respectively. Comparing the computation time, we see that the algorithms can be ordered from fastest to slowest as VIH-1, SBEH, VIH-5, VIH-10, ABEH, SA, BEH and MIP. In terms of solution quality, the algorithms can be ordered from best to worst as MIP, BEH, ABEH, SA, SBEH, VIH-10, VIH-5 and VIH-1. In the last table we report in what proportion of cases an algorithm got a better solution in less time than another algorithm. Here we see that BEH dominated MIP in 7% of the cases, while the other way around was only 1%. If we combine this with the unpredictability of the MIP running time, we would recommend the BEH in the slowest category. In deciding between SA, ABEH and VIH-10, we observe that ABEH dominated VIH-10 in 46% of the cases, while the other way around is just 3%. At the same time, ABEH beats SA in 59% of the cases, as opposed to just 1% the other way around. Therefore, we conclude that ABEH has the best performance among the balanced algorithms. Finally, for the fast algorithms SBEH dominates VIH-5 in 57% of the cases. If we combine this with the unreliability of the VIH-1, it becomes clear that SBEH should be used to obtain approximate solutions quickly.

5.4 Extensions

MDP solving versus value function estimation

For our heuristics we discuss two methods of evaluating the change in cost due to exchanging buffer between two legs. We can choose between the exact method of solving an MDP, for which we use linear programming, and an estimate based on the value function. We name the former LP and the latter VE (value estimation) It is of interest to know how good of an estimate is made by the VE method. To investigate this we generated 1000 random allocations with a random 1-exchange for each allocation and computed the cost change using both methods. As a performance measure we use the absolute value of the percentual deviation of VE from LP. In Table 8 some key statistics on this analysis are reported and in Figure 2 a histogram of the performance measure mentioned above is shown.

Mean	527%
Mean (only values <500%)	118%
Coefficient of variation (% of mean)	474%
Median	102%
Mode	20%
Sign right (% of cases)	71%

Table 8. Statistics on absolute value of percentual deviation of cost change estimates from true cost change

One can immediately see that the estimates are quite far off the mark. On average, they deviate 527% from the real values. As this is heavily influenced by some outliers, which will be discussed later, we also show the mean where we only consider values that are at most 500% off. Even in this truncated sample the estimates miss the mark by over 100% on average.

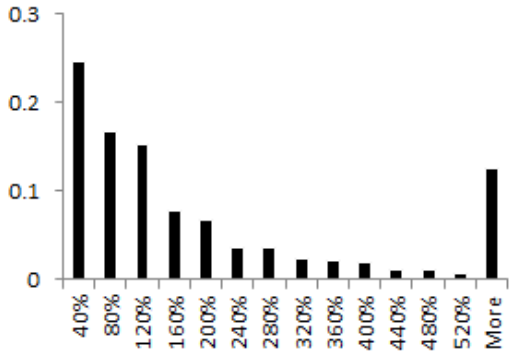

Figure 2. Histogram of comparison performance measure

Another finding is that the coefficient of variation (standard deviation divided by mean) is almost 500%. Not only is the deviation of the estimate from the mean large, but it is also highly random. Because of the large number of outliers, as can also be seen in the histogram, we also report some more robust statistics. The most interesting of these is that the estimate has the right sign in just 71% of the cases. It is questionable whether the exchanges that are optimal under the value estimates are also good in reality. To investigate this further, we display a number of scatter plots. In Figure 3 the exact values are on the vertical axis, while the estimates are on the horizontal axis. One would prefer these to lie on a $45°$line. There is some clustering around a line through the origin, but a large number of observations are quite far off. Regressing the estimates on the exact values in the model $y = \beta x + \varepsilon$ gives $\hat{\beta} = 0.56$ with a standard error of 0.075. Testing $H_0 : \beta = 1$ gives a t-value of 5.87, clearly rejecting the null hypothesis. There is no evidence to suggest that the values lie on a $45°$line.

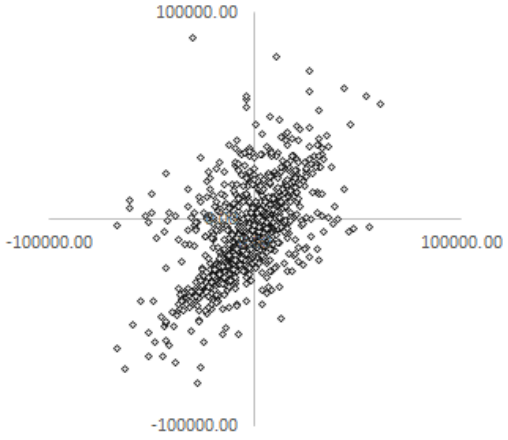

Figure 3. Scatterplot LP (vertical) vs VE (horizontal)

A last point of interest is whether anything can be said about when the estimates are bad and when they are better. For this purpose we show Figure 4, in which we show the performance measures on the vertical axis and the absolute exact values on the horizontal axis. Here one sees that the performance measure tends to be very high for low exact values. This is partly due to that the exact value is in the denominator: if it is low, the measure is higher. On the other hand this seems to indicate that the deviation is not proportional to the exact value. Put differently, it seems that its just as likely to be off by 1,000 if the LP estimate is 10 as when it is 10,000. Figure 5 shows the absolute values of the difference between the estimate and the exact value on the vertical axis against the exact value on the horizontal axis. If the difference was proportional to the exact value, these should lie on a $45°$line trough the origin. They seem quite randomly distributed though, leading to the conclusion that the deviation of the estimate from the exact value and the exact value are not related in a straightforward manner. This means that the error is especially large for small cost reductions. Summarizing: the value estimates tend to be off quite far from the exact values and it is not advisable to rely on them too much. This is another reason to use SBEH for quick problem solving, instead of VIH-1 or VIH-5.

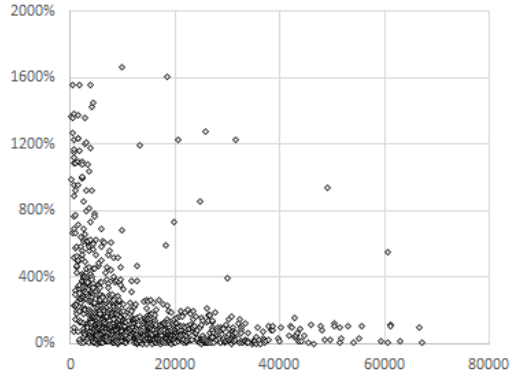

Figure 4. Scatterplot performance measure (vertical) vs absolute LP values (horizontal)

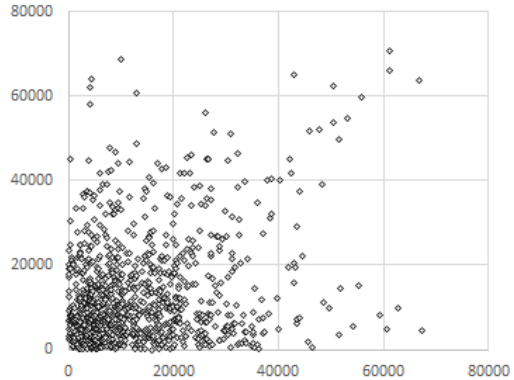

Figure 5. Scatterplot absolute deviation VE from LP (vertical) vs absolute LP values (horizontal)

Cost as function of total buffer time

In their paper, Mulder and Dekker prove that the total cost for a route is non-decreasing in the total amount of buffer that can be allocated. It seems likely that the cost reduction due to an additional unit of buffer time is decreasing in the amount of buffer time already allocated. To investigate this further we computed the total expected cost for the case presented earlier, without disturbance in the delays. The cost was computed using the SBEH, yielding Figure 6.

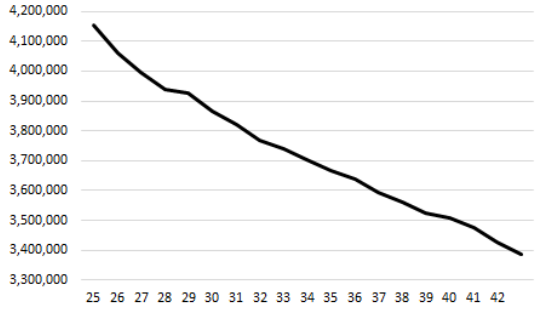

Figure 6. Expected cost as a function of available buffer time

One can observe that the total cost is indeed strictly decreasing in the amount of available buffer. However, one cannot conclude convexity based on this diagram. It seems that for most of the interval the shadow price of buffer time decreases only slightly. Another finding is that on this route, at least 25 units of buffer time are required with our delay distribution. If less buffer time is available, the delay will frequently rise above the delay cap and lead to very high costs. This also has to do with that we have defined the schedule as one were ships already sail at their top speed. This makes buffer time the only way delays can be compensated.

Mean estimation for right-censored computation times

When drawing conclusions about the computation times, we are interested in the random variable T, the computation time of an algorithm on some instance. However, the algorithms are cut off if they run for longer than some time t^*. Therefore, instead of T, we observe another random variable X, defined by Equation 34.

$$X = \begin{cases} T & \text{if } T \leq t^* \\ t^* & \text{if } T > t^* \end{cases} \tag{34}$$

When doing computational experiments, we observe a sequence of realizations for X. By taking the sample mean, we obtain an unbiased and consistent estimate of $E[X]$. However, this estimator is biased and inconsistent for $E[T]$, as the following derivation shows that $E[T] > E[X]$.

$$\begin{aligned} E[X] =& P(T \leq t^*)E[X|T \leq t^*] + P(T > t^*)E[X|T > t^*] \\ =& P(T \leq t^*)E[T|T \leq t^*] + P(T > t^*)t^* \\ <& P(T \leq t^*)E[T|T \leq t^*] + P(T > t^*)E[T|T > t^*] \\ =& E[T] \end{aligned}$$

To obtain a good estimate of $E[T]$, we have to incorporate information about the censoring in it. Several approaches to dealing with censored data exist. A commonly used one is the non-parametric Kaplan-Meier estimator of the survival function (see for example Borgan, 2005). This method actually assumes a more general situation, in which t^* is a random variable too, which is not the case. Additionally, estimating the mean using the Kaplan-Meier estimator is not straightforward. Therefore we choose to use a maximum likelihood approach instead. A downside of this approach is that we need to postulate a distribution for T.

Say that T has CDF $F(t;\boldsymbol{\beta}) = P(T < t)$ and pdf $f(t;\boldsymbol{\beta}) = \frac{d}{dt}F(t)$, both dependent on a vector of parameters $\boldsymbol{\beta}$. Furthermore, say a one-to-one function exists such that $E[T] = g(\boldsymbol{\beta})$. The invariance property of maximum likelihood estimates implies that by obtaining a maximum likelihood estimator $\hat{\boldsymbol{\beta}}$ for $\boldsymbol{\beta}$, we obtain a maximum likelihood estimator $g(\boldsymbol{\beta})$ for $E[T]$. This estimator is $\sqrt{n}-$consistent and asymptotically efficient. Considering a random sample $x_1, \ldots, x_n$, the maximum likelihood estimate for $\boldsymbol{\beta}$ is given by Equation 35.

$$\begin{aligned} \mathcal{L}(\boldsymbol{\beta}) &= \sum_{i:\, x_i < t^*} \log(f(x_i;\boldsymbol{\beta})) + \sum_{i:\, x_i = t^*} \log(1 - F(t^*;\boldsymbol{\beta})) \\ \hat{\boldsymbol{\beta}} &= \arg\sup_{\boldsymbol{\beta}} \{\mathcal{L}(\boldsymbol{\beta})\} \end{aligned} \tag{35}$$

It is highly unlikely that the MIP computation times follow some standard distribution. We will look for a distribution that resembles reality as close as possible. Among all possible distributions, we consider the Exponential, Weibull and Gamma distributions. The first two are commonly applied to lifetime modeling and the last is relevant if the MIP computation time consists of a series of $i.i.d.$ exponential random variables. In Figure 7 we compare the computation times with the pdf's of the relevant distributions, with the parameters estimated using Equation 35. We conclude that a Weibull distribution

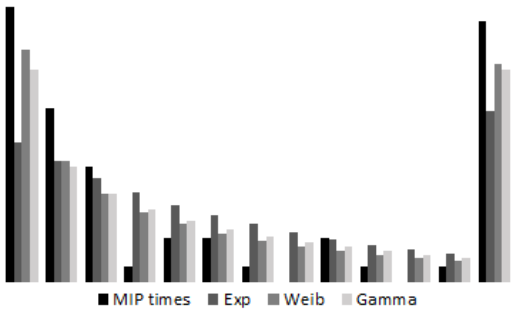

Figure 7. Frequency diagram for MIP computation times with pdf's

with parameters $(\hat{\lambda} = 1969,\ \hat{k} = 0.71)$ fits the data best. The mean of a Weibull distributed variable is given by $\lambda\Gamma(1 + \frac{1}{k})$. This gives an estimated mean of 2444. As expected, the mean estimate that we find in this way is significantly higher than the sample mean from the uncensored data.

6. Conclusion

In this thesis we looked at the problem of incorporating delays into an existing vehicle schedule by allocating buffer time to the various legs of the route and by deciding on recovery actions that should be taken if delay has been incurred. For this problem a mixed integer programming formulation and several heuristics had already been proposed by Mulder and Dekker (2016), whose paper we aimed to replicate and extend. They also discussed solving the problem of choosing recovery actions if a buffer allocation is given as a Markov Decision Process (MDP). Of the heuristics that they proposed, we considered two that evaluated the cost reduction due to exchanging buffer time between two legs for all possible exchanges and made the most profitable one. The cost reduction was computed either by solving an MDP (called the Buffer Exchange Heuristic, BEH) or by estimating it using the value function (called the Value Iteration Heuristic, VIH). They recommended the VIH, because it gave reasonable solutions in little time.

We proposed several new heuristics for this problem. The simplest was to use multiple starting points in the VIH, to decrease the probability of ending up in a bad local minimum. Another new method, called the Adapted Buffer Exchange Heuristic (ABEH), was not to compute the cost reduction for all exchanges in each iteration, but computing it once per so many iterations and considering only the exchanges which were found to give a cost decrease in the next few iterations, such that the cost reduction had to be computed for less exchanges. Two other methods that also reduced the number of exchanges to be evaluated were the Stochastic Buffer Exchange Heuristic (SBEH) that draws random exchanges and only evaluates the cost change for these exchanges and Simulated Annealing (SA) that did the same, but also allows exchanges that lead to a cost increase with a positive probability. This probability decreases as the algorithm progresses.

The main goal of this thesis was to recommend good methods for solving the problem sketched above. For this purpose, a computational study was done in which 70 test cases, based on the ME1 route of the Maersk Line Network, were generated. All methods were used separately to solve each test case and the computation time and final objective value were recorded. When comparing these, we found that the algorithms could roughly be divided into three categories. The MIP and BEH are the slowest, but give high quality solutions. Of these two, the BEH is to be recommended, as it is faster than the MIP and gives solutions that are almost always as good. Furthermore, the running time for the MIP is unpredictable and can be very long, while the computation time for the BEH is much more stable. The ABEH, SA and VIH with 10 starting points are the most balanced algorithms, giving good solutions in a reasonable time frame. Of these, the ABEH clearly dominates both SA and VIH in terms of computation time as well as solution quality. Therefore we recommend the ABEH for cases where a good solution is required, but the problem is large or time constraints are relevant. Finally, SBEH and VIH with 1 or 5 starting points fall in the fastest category. VIH was shown to be quite unreliable, giving costs of up to 25% more than the MIP solution in some cases. VIH-5 is also slower than SBEH. Therefore, in case the problem needs to be solved as quick as possible or the problem instance is very large, the SBEH should be used. An analysis of the quality of the value function estimates used for the VIH showed that these estimates are quite poor in general, providing another argument for the SBEH as opposed to the VIH.

References

[1] Abramowitz, M., Stegun, I. (1972). "§24.1.3. Stirling Numbers of the First Kind". Handbook of Mathematical Functions with Formulas, *Graphs, and Mathematical Tables (9th ed.)*. New York: Dover. p. 824.

[2] Borgan, Ø. (2005). Kaplan–Meier Estimator. *Encyclopedia of biostatistics*.

[3] Brouer, B. D., Dirksen, J., Pisinger, D., Plum, C. E., & Vaaben, B. (2013). The Vessel Schedule Recovery Problem (VSRP)–A MIP model for handling disruptions in liner shipping. *European Journal of Operational Research*, 224(2), 362-374.

[4] Brouer, B. D., Alvarez, J. F., Plum, C. E., Pisinger, D., & Sigurd, M. M. (2013). A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2), 281-312.

[5] Brinkhuis, J., & Tikhomirov, V. (2011). Optimization: insights and applications. Princeton University Press.

[6] Eglese, R. W. (1990). Simulated annealing: a tool for operational research. *European journal of operational research*, 46(3), 271-281.

[7] Garey, M. R., & Johnson, D. S. (1978). "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, 25(3), 499-508.

[8] Garey, M. R. & Johnson, D. S. (1979). Computers and Intractability, A Guide to the Theory of NP-Completeness. Freeman & Co., San Francisco

[9] Johnson, D. S., Papadimitriou, C. H., & Yannakakis, M. (1988). How easy is local search?. *Journal of computer and system sciences*, 37(1), 79-100.

[10] Kirkpatrick, S., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671-680.

[11] Lenstra, J. K. (1997). Local search in combinatorial optimization. Princeton University Press.

[12] Li, C., Qi, X., & Lee, C. Y. (2015). Disruption recovery for a vessel in liner shipping. *Transportation Science*, 49(4), 900-921.

[13] Mulder, J & Dekker, R. (2016). Designing robust liner shipping schedules: Optimizing recovery actions and buffer times, *working paper*.

[14] Naumann, M., Suhl, L., & Kramkowski, S. (2011). A stochastic programming approach for robust vehicle scheduling in public bus transport. *Procedia-Social and Behavioral Sciences*, 20, 826-835.

[15] Notteboom, T.E. (2006). The time factor in liner shipping services. *Maritime Economics & Logistics*, 8(1), 19-39.

[16] Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). The complexity of Markov decision processes. *Mathematics of operations research*, 12(3), 441-450.

[17] Puterman, M. L. (2014). Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons.

[18] Qi, X. (2015). Disruption Management for Liner Shipping. *In Handbook of Ocean Container Transport Logistics* (pp. 231-249). Springer International Publishing.

[19] Ross, S. M. (2014). Introduction to probability models. Academic press.

[20] SeaRates. 2015. Reference guide, port to port distance. *https://www.searates.com/reference/portdistance*. Acessed: 19 October 2015

[21] Thengvall, B. G., Bard, J. F., & Yu, G. (2000). Balancing user preferences for aircraft schedule recovery during irregular operations. *IIE Transactions*, 32(3), 181-193.

[22] Walker, C. G., Snowdon, J. N., & Ryan, D. M. (2005). Simultaneous disruption recovery of a train timetable and crew roster in real time. *Computers & Operations Research*, 32(8), 2077-2094.

[23] Wang, S., & Meng, Q. (2012). Robust schedule design for liner shipping services. *Transportation Research Part E: Logistics and Transportation Review*, 48(6), 1093-1106.

[24] White, C. C., & White, D. J. (1989). Markov decision processes. *European Journal of Operational Research*, 39(1), 1-16.