# Tramp Ship Scheduling with Speed Decisions: A Tabu Search Approach

Marc Kuo – 292131

Main Supervisor: Prof. Dr. Ir. U. Kaymak
Second Supervisor: Prof. Dr. Ir. R. Dekker

Internship at Capgemini
Advanced planning and scheduling
Supervisor: Dr. R. Schilham

December 8, 2009

# ABSTRACT

Slow steaming has been a known method of saving costs anecdotally, but for the first time, this paper quantifies this concept explicitly on ship scheduling.

This study analyzes the potential of incorporating speed decisions within the tramp ship scheduling problem. Two different mathematical models, one with and one without speed decisions, were devised. We developed a tabu search algorithm with an efficient embedded speed optimization heuristic, and optimized both decision variables simultaneously.

An extensive computational study was performed, with various simulated real-life test cases. On 71 out of the 72 test cases, the Tramp Ship Scheduling model with Speed Decisions outperformed the general Tramp Ship Scheduling model with great significance.

The new model may enable a shipping company to save tens of millions of dollars a year on fuel consumption. Bearing the dynamic tramp industry in mind, the tool may run for as long that is allowed to improve solution quality, but will also yield very good solutions in very limited time.



Keywords: Tramp Ship Scheduling, tabu search, Speed Optimization

CONTENTS


*1. Introduction* . . . . . 5
1.1 Relevance . . . . . 5
1.2 Research Question . . . . . 7
1.3 Methodology . . . . . 8
1.4 Thesis structure . . . . . 9

*2. Maritime Scheduling* . . . . . 10
2.1 Tramp scheduling . . . . . 10
2.2 Cruising speed . . . . . 12
2.3 Solution approaches . . . . . 13
2.4 Summary . . . . . 15

*3. The Tramp Ship Scheduling Problem: A Mathematical Model* . . . . . 16
3.1 Notation . . . . . 16
3.2 General tramp ship scheduling model . . . . . 19
3.3 Tramp ship scheduling model with speed decisions . . . . . 20
3.4 Tramp ship scheduling model with speed decisions and soft time windows . . . . . 21
3.5 Summary . . . . . 22

*4. Tabu search* . . . . . 24
4.1 Introduction to Tabu Search . . . . . 25
4.1.1 Neighborhood structure . . . . . 25
4.1.2 Short term memory . . . . . 27
4.1.3 Long term memory . . . . . 27
4.2 Tabu Search Algorithms for the Vehicle Routing Problem . . . . 28
4.2.1 Taburoute . . . . . 28
4.2.2 Taillard's Algorithm . . . . . 29
4.2.3 Adaptive Memory Procedure . . . . . 29
4.2.4 Granular Tabu Search . . . . . 30
4.2.5 Reactive Tabu Search . . . . . 30
4.3 Summary . . . . . 31

*5. Tramp Tabu Search* . . . . . 32
5.1 Initial feasible solution . . . . . 32
5.2 Neighborhood structure . . . . . 33

5.2.1 Paired insertion move . . . 34
5.2.2 Swap move . . . 36
5.2.3 Neighborhood experiments . . . 37
5.3 Tabu List . . . 38
5.4 Overview of the TTS algorithm . . . 43
5.5 Embedded Speed Optimization Heuristic . . . 43
5.5.1 Dual optimization . . . 43
5.5.2 Speed optimization issues . . . 45
5.5.3 Speed optimization heuristic . . . 46
5.6 Summary . . . 47

*6. Quantitative Analysis of Slow Steaming Opportunities* . . . 49
6.1 Benchmark study . . . 49
6.2 Verification test cases . . . 50
6.2.1 Test case: Spot charter . . . 52
6.2.2 Test case: Speed Decisions . . . 53
6.2.3 Test case: Fleet . . . 54
6.3 Quantitative effects of Speed Decisions . . . 55
6.3.1 Experimental setup . . . 55
6.3.2 Computational results . . . 59
6.3.3 Further analysis . . . 62
6.3.4 Summary . . . 67

*7. Conclusion and Discussion* . . . 68

# 1. INTRODUCTION

## *1.1 Relevance*

Seaborne shipping is the most important mode of transport for international trade. An estimated 80 per cent of world merchandize trade by volume is carried by sea [35]. Other common modes of freight transportation are by truck, train, aircraft and pipeline. Compared to these, ships are far superior for moving large volumes over long distances. Consequently, the shipping industry has a monopoly on intercontinental transport.

Due to the trend of an increasing standard of living, developing economies and globalization, with more international trade as a result, the sector of maritime transport has shown an enormous growth and is likely to grow even further. Table 1.1 shows the magnitude of the increasing trend of seaborne shipping worldwide from 1970 on [35].

Ships require an enormous capital investment and very high daily operating costs. Investment in a ship may range in the millions and operating costs in the thousands a day, with bigger ships tenfold. Good planning or scheduling is more than interesting; it is of economical essence in this increasingly competitive era.

In practice, little or no advanced computer based techniques are applied for making fleet schedules. Conventionally, these schedules are manually made using pen and paper, perhaps visualized in a spreadsheet, and are based on sheer experience [14]. Manufacturing companies have grown bigger and consolidated their demand for sea transport, resulting in increased market power for the cargo owners. To compensate for this, many small shipping companies have merged in the last decade, or entered in pooling efforts. As a consequence, fleet sizes increase, available information explodes, and conventional manual planning will

| Year | Total Cargo (million tons) | Total ton-miles (billions) |
|---|---|---|
| 1970 | 2566 | 10654 |
| 1980 | 3704 | 16777 |
| 1990 | 4008 | 17121 |
| 2000 | 5984 | 23693 |
| 2006 | 7652 | 31447 |
| 2007 | 8022 | 32932 |

*Tab. 1.1:* Growing trend of maritime transport

no longer suffice [5].

Despite these compelling figures, the science of maritime transport is not nearly as explored in literature as compared to the other modes of transport, such as aircraft and truck. Reasons for this lack mentioned in [10] include low visibility. Ships are the least visible comparing to trucks, aircrafts and trains. Furthermore, most cargo is moved by truck over short distances. There are many big organizations that own a large truck fleet that sponsor research in the planning of their fleet; such organizations are rather rare in the shipping industry.

Three modes of operation in the shipping industry are distinguished in [25]: *liner, industrial,* and *tramp* operations. Liners can be compared to a bus line, and operate according to a published schedule. The demand is dependant on the schedule or itinerary of the liner. Industrial shippers usually own the cargo and the ships themselves and are vertically integrated in the company. Tramp ships are like a taxi, and go there where is demand for their service. The business of a tramp ship is usually made up of the known demand in *contracts of affreightment* and uncertain demand in *spot markets*. Contracts of affreightment are long-term contracts that specify the amount of cargo that need to be transported between specified ports within a specified time window. A fixed price per unit of cargo is agreed upon. Meanwhile a tramp ship tries to maximize its profits from optional cargos that occur in the spot markets if feasible. Whereas both liner and tramp operators maximize their profit per time unit, industrial shippers minimize the cost of shipping, resulting in a slightly different problem approach.

Looking at a recent review by [9], summarizing studies on ship routing and scheduling of the past decade, we see that most contributions are in industrial shipping. Many large manufacturing companies were vertically integrated and had their own fleet for transport. Since the trend nowadays is to focus on core competencies, more and more companies outsource their transportation division to independent shipping companies. Many small tramp shipping industries started to merge. In parallel with the rise of tramp shipping companies is the increasing fleet size of these shipping companies, hence the increasing complexity of planning problems and the need for advanced decision support systems. Additionally, as compared to industrial shipping where all cargo is required to be shipped, tramp shipping companies are given the possibility to ignore some demand to maximize profits. Therefore, the solution space is even more complex, and decision support systems have an even higher added value. This thesis contributes to the research in the tramp shipping industry.

Like other planning problems, maritime transport planning problems can be classified in three overlapping levels, i.e. *strategic, tactical,* and *operational* problems. Among strategic problems one can think of network design problems or fleet size/mix problems. These problems have the highest impact and the longest planning horizon. Some tactical problems, among others, include ship routing and scheduling, crane scheduling and fleet deployment. These problems have less impact and their problems can be dependent on decisions on the strategic level; the nature of a ship routing and scheduling problem is dependent on the network design available. Operational problems are for example speed

selection and environmental routing, and have even lower impact and smaller planning horizon. Similarly, speed selection problems have a direct impact on the tactical level of scheduling, which is the topic of this thesis.

Regarding speed selection, according to [27], a ship's fuel consumption is directly related to the third power of the speed. This means that by slowing down 20% will reduce fuel consumption by approximately 50%. Significant operational cost savings may be achieved by steaming optimally as opposed to nominally. However, opportunity costs may be higher than the fuel savings achieved by slow steaming. If the cargo's daily interest is higher than the daily fuel cost savings, it makes no sense to delay the transportation [31]. Alternatively, when considering soft time-windows as in [13], penalty of a late delivery may also surpass fuel savings. On the other hand, when the ship may be used as a floating storage and keeping the cargo at sea is cheaper than on land, sailing at minimum speed is obvious. Other obvious examples one could think of is when a ship is going to arrive at the port when it is not working, e.g. during weekends or holidays. Slowing down and delaying its arrival will save fuel costs without any other tradeoff. On a bigger scale, when a shipping company owns a big fleet which burns fuel at a daily rate of millions of dollars, sailing optimally by deciding on cruising speed may result in savings of millions of dollars annually.

This research discusses a planning problem on both the operational and the tactical level. We discuss the problem in the tramp industry of deciding on cruising speed on the operational level, while solving for the scheduling problem on the tactical level. To this end we propose a mathematical model in which sailing speed is explicitly taken into account. The objective is to maximize profits, which means maximizing revenues and minimizing operation costs simultaneously. Revenues are created from transporting contracted cargoes and selecting the most profitable optional cargoes in the spot market. Also, spot charters may be deployed for single voyages to account for insufficient capacity. The main operating cost is bunker fuel, which is directly related to the sailing route, i.e. schedule, as well as the cruising speeds. Alternating cruising speeds of ships does not go without consequences for the scheduling problem on the tactical level. Conventionally, ship routing and scheduling problems are dealt with given a specified cruising speed of ships, usually the maximum speed. Varying in cruising speed not only changes the associated sailing costs, but also the arrival times, with all the consequences for the scheduling problem. The search space is larger and would require more computational effort to find (sub)optimal solutions.

## 1.2 Research Question

The objective of this research is to analyze the incorporation of sailing speed decisions in the tramp scheduling problem. Taking decisions regarding cruising speed into account will enlarge the solution space, making the scheduling problem much more complex. The following research question can be formulated:

- What are the consequences of incorporating cruising speed decisions in the tramp shipping scheduling problem?

The following sub-questions are derived from the main research question:

- How do we model a realistic tramp scheduling problem?
- How do we solve the formulated model?
- How do we incorporate speed decisions in our algorithm?
- How do we generate realistic test cases to test our models on?
- Is the new approach computationally feasible for the dynamic tramp industry?

## 1.3 Methodology

In order to answer our main research question, we implemented two different models. One model solves the tramp shipping scheduling problem with and the other without cruising speed decisions. The performance of both models are assessed on various realistic test cases, and compared with each other on solution quality and computational effort. From the results, we concluded whether the benefits of a better solution of the model with speed decisions, outweigh the drawbacks of increased computational effort.

Before implementing our models, we first performed a literature study in the more covered realm of Vehicle Routing Problems. We first developed a tabu search algorithm for the Pickup and Delivery Problem with Time Windows, and verified the algorithm with benchmark data and tuned it to perform as good as possible[1]. With this strong foundation, we extended the algorithm to solve for the tramp ship scheduling problems, resulting in the general Tramp Ship Scheduling (TSS) model. Then, we devised an efficient heuristic and embed it into the tabu search, which optimizes sailing speeds simultaneous to schedule optimization, resulting in a new model: Tramp Ship Scheduling with Speed Decisions (TSS-SD).

The models are first tested on arbitrarily small test cases, before both models are run on practically sized data sets. Numerous different test cases are generated using data gathered from the shipping industry. Additionally, numerous parameter tuning experiments are put in place while developing our algorithm.

The following steps describe the methodology we used for this research.

- Firstly, a literature study is performed in the fields of maritime transport, tramp scheduling, and search methodologies regarding routing and scheduling problems (including Vehicle Routing Problems).

[1] Good, here, meaning striking a good balance between computational effort and solution quality. This is while bearing in mind the dynamic tramp industry, where new schedules need to be generated frequently and quickly.

- Then, we formulated a general mathematical model for the tramp shipping problem, based on the models of [6, 7].
- This model is solved using tabu search, implemented in Java.
- We extended the mathematical model by incorporating cruising speed decisions.
- This new model uses a novel speed optimization heuristic embedded in the tabu search.
- Performance of the models are assessed on numerous realistic test cases.

## 1.4 Thesis structure

The rest of the thesis is structured as follows. Chapter 2 explores literature in ship routing and scheduling, with the emphasis on the tramp industry. Different solution approaches are described and the linkage with VRP solution approaches is made. Chapter 3 gives the formal mathematical formulations of the models we propose that account for cruising speed decisions, build on a general tramp scheduling model. Then we give a short introduction in the tabu search algorithm, and describe our proposed solution approach in Chapter 4. This chapter also includes the description of the embedded speed optimization heuristic for our TSS-SD model. The experimental setup and results of the computational study are presented in Chapter 6, and the conclusions drawn from them in Chapter 7.

# 2. MARITIME SCHEDULING

The literature of maritime transport has seen a growth in interest during the last decades, with the first survey of [32] on ship routing and scheduling dating from 1983, tracing papers back in the 1950s. Ten years later, a second survey [33] for the period of 1982-1992, and most recently another survey [9] for the OR work in ship routing and scheduling in the last decade for the period of 1993-2003. The first two review papers included about 40 references, whereas the most recent one almost doubled the number, showing increasing interest in this field of research. However, most publications are regarding the industrial shipping industry; little work was done in tramp ship scheduling.

Firstly, we give a summary of the few papers in tramp scheduling literature in Section 2.1. Here, only those that strictly studied the tramp industry are considered. Then, in Section 2.2, papers that address the problem of ship cruising speed are presented. Finally, Section 2.3 presents some solution approaches that have been applied in the literature to the ship scheduling problem. This section also makes the link to the well-studied Vehicle Routing Problem, and tabu search heuristics that have been applied successfully to this problem.

## *2.1 Tramp scheduling*

Tramp ships operate like a taxi and go there where is demand. Tramp shipping companies may engage in *contracts of affreightment* which specify the amount of cargo that need to be transported between specified ports in a specified time window. Beside these contracted cargoes that require to be shipped, the shipping company tries to maximize profits in serving demand that occurs in the *spot markets*. Furthermore, when the fleet has insufficient capacity to meet all demand, the shipping company can consider serving the additional cargo by using spot charters. These are ships from an external party chartered for a single voyage and may be deployed when profitable.

The main difference with industrial shipping companies, where the fleet and cargo is owned by one company, is that the objective of a tramp shipping company is to maximize profits as opposed to minimize costs. A tramp shipping company is given flexibility to make choices to ship or not to ship spot cargoes. A ship can make more money by shipping the most profitable cargoes only and rejecting those that pay little. Industrial ships do not have this flexibility as all cargo is required to be shipped. This additional flexibility in the tramp industry means that the solution space is bigger, and schedule optimization will have

a bigger financial impact. Nonetheless, most research has been in industrial shipping for the reasons mentioned in the introduction and [9].

Due to the little amount of studies in tramp scheduling, it is often taken together with industrial ship scheduling literature. However, the different industries have different objectives, which results in a different problem; a distinguishment in literature should be in place. Below we discuss a few studies in tramp ship scheduling.

*Full shiploads* A typical tramp ship scheduling problem was described in [2]. They assume, however, full shiploads, as only one cargo is carried at the time. The problem at hand is to assign sequences of cargos to each ship optimally; the objective is to maximize revenues of optional cargoes minus the voyage and fuel costs. All contracted cargoes must be shipped though.

Since spot market demand occurs after a schedule has been made, frequent rescheduling is required. A major advantage of using advanced computerized scheduling is that new schedules can be created timely.

Since it is unsure when new demand occurs, [2] prepares for these opportunities by concentrating scheduled cargoes to small ships with restricted cargo capabilities, leaving more flexibility open. They introduced a so called 'time value' for each ship, which is a daily premium for idle time after completion of the last scheduled cargo. This premium varies between ships distinguishing in size and cargo capabilities, which makes some ships more preferable to keep idle.

[2] was the first one to describe a tramp scheduling problem and used a Dantzig-Wolfe decomposition solution approach. The master problem is a linear relaxation of a Set Partitioning Problem, and the subproblems are simple shortest path problems. For more details of this solution approach, refer to [2]. This approach has been used later in numerous other studies.

*Spot charters* A heterogeneous fleet consist of ships that may carry multiple cargoes simultaneously and cargoes can be loaded irrespective of the type of cargo already on board. When the fleet has insufficient capacity to satisfy all demand, some cargoes may be serviced by spot charters. Spot charters are external ships chartered for a single voyage that can be deployed when the capacity of the available fleet is insufficient to meet all contracted cargoes during the planning horizon. For ship scheduling models, the availability of a spot charter is out of concern. The assumption is made that at all times, there is a spot charter available. Among tramp scheduling studies that incorporated spot charters in their model are for example [14, 6, 5, 7].

*Flexible cargo sizes* The assumption of fixed cargo quantities is relaxed later by an extension in [6, 7]. They present a MP-model of a multi-ship pickup and delivery problem with time windows and flexible cargo sizes. The motivation for flexible cargo sizes is that for real long-term contracts of affreightment, the sizes of cargoes may often be flexible. Cargo quantities are often given in an

interval, which is a so called 'freedom of contract'. Two main categories are the MOLOO (More Or Less Owners Option) contracts and the MOLCO (More Or Less Carriers Option) contracts. Both contracts specify an interval for the quantity, but the latter gives the shipping company, as opposed to the cargo owner, the right to specify the actual quantity upon loading. The associated revenue of carrying a cargo depends on the contract. Wider time windows and large quantity intervals will be cheaper, hence generate less revenue for the shipping company. Some advantages of utilizing cargo quantity flexibility, mentioned by [23], include reducing cargo quantities in order to serve additional spot cargoes, and to increase total load of cargoes for more revenues.

A mathematical graph representation model is described by [6, 7]; They formulated a set partitioning approach to solve this problem where they generate columns *a priori*. This is done by enumerating all feasible visit sequences and solving an LP problem for all sequences to optimize the quantities for each visit sequence. To increase computational efficiency they only call upon a commercial LP solver if the visit sequence at hand is difficult to optimize by hand. They show in a computational study that an increased flexibility in quantities may lead to increased profits.

## *2.2 Cruising speed*

Decisions regarding cruising speed of ships may have a significant effect on operational costs, i.e. fuel savings. Savings in fuel consumption also result in lowering of the carbon footprint. According to [27], fuel consumption is directly related to the third power of the speed. This means that by slowing down 20% will reduce fuel consumption by approximately 50%. Although this relationship is approximated empirically, it remains useful and accurate. However, a ship's cruising speed is bounded by a minimum speed at which proper steering is still assured.

With high fuel prices like those during the 1970s, it may very well be the highest operating cost and incentive is there to consider slow steaming. To this end, [31] has studied the effect of cruising speed of ships, and devised three models of cruising speed optimization. The trade-off considered is the fuel savings achieved by cruising slow, and the loss of additional revenue on the other hand in the form of chartering out ships, i.e. the opportunity costs.

[8], and later again in [3], addressed the problem of deciding on an optimal cruising speed inherently with creating the optimal ship schedule for the tramp shipping industry. Both approached the problem with an Elastic Set Partitioning Problem solution method. However, since the assumption of hard delivery time windows is made, only decisions regarding cruising speed is made on ballast (empty) legs. For loaded trips, the speed is implicitly determined by the loading and discharging dates of the cargo.

[13] considers soft time windows, as opposed to the conventional hard time windows. By allowing time window violations for some customers, at an appropriate inconvenience cost, better schedules and significant reductions in trans-

portation costs can be achieved. It allows for more flexibility in optimal cruising speed determination. They used a set partitioning formulation in which they enumerate all promising candidate ship routes a priori. For each route (subset of nodes), the optimal schedules are calculated as a Traveling Salesman Problem minimizing their operating and inconvenience costs. Here, ship's speed, hence fuel consumption or operating costs, depends on the timing of the stops on the route. To this end, time is made discreet and a new network is generated by duplicating the nodes to represent different arrival times. The shortest path is then found by dynamic programming. These optimal schedules are then inserted in the set partitioning problem.

When considering alternating cruising speed and making schedules simultaneously, one should take beside fuel consumption, also port operation times into consideration. Arriving at a closed port in the middle of the night is of no use; more fuel could have been saved by sailing slower. On the other hand, when service will continue once started even after closing time, it might be worthwhile to save time by sailing faster to get there right before the port closes [10].

## 2.3 Solution approaches

*Column generation* As ship scheduling problems are tightly constrained, it is possible to enumerate all feasible schedules for all cargo set-ship combinations. Once these schedules (columns) are generated, they can be inserted in a set partitioning problem. The advantage of this column generation approach is that all constraints regarding feasibility of schedules can be taken into account while generating the schedules. When a problem is tightly constrained, this results in a reasonable amount of decision variables, hence speeding up the optimization process.

Most studies in the ship routing and scheduling literature apply the set-partitioning solution approach or column generation of some sort [2, 8, 22, 3, 13, 6, 7].

In [22] a simple generalized set-packing model is formulated for a tramp ship scheduling problem. They present an algorithm for generating all binary decision variables (columns), i.e. all feasible schedules for each ship, a priori. A Dantzig-Wolfe procedure is presented in [7] to the tramp ship scheduling problem with flexible cargo sizes. They showed that their approach is faster than the a priori generation of column; large and loosely constrained problems, that were intractable for a priori column generation, have been tackled.

*Heuristics* Some studies have tried heuristics to speed up the process, as column generation may require high computational effort and may be unsuitable for very large problems. [5] applied a multi-start local search heuristic. The basic outline of this approach is to create a large number of partly biased randomized initial solutions, that comprise of solutions with quality and diversity. Next, a selection of initial solutions are improved by a quick local search heuristic. Then, from these improved solutions, another selection is made and an extended local

search heuristic is performed on them. A computational study is performed comparing the results and efficiency of this algorithm with a set-partitioning approach that enumerates all columns a priori. For semi-large cases, consisting of 18 cargoes and 6 ships, an optimal solution is found with the set-partitioning approach in 100 minutes, versus 4 seconds using the multi-start local search heuristic. This enormous gain in efficiency, only to trade-off lower than 1 per cent of solution quality, is very practical for the dynamic tramp industry.

In [14], the development of a practical Decision Support System is described. As they focus on fast practical solutions, as opposed to theoretical (near-)optimal solutions, they apply simple heuristics. First, an insertion heuristic is applied to insert all cargoes sequentially into the work schedule of a vessel to obtain an initial solution. Then, the initial solution is improved using a hybrid local search algorithm.

[23] tackled the tramp ship scheduling problem with flexible cargo sizes, but used a tabu search approach algorithm embedded with a specialized heuristic for deciding the optimal cargo quantities for each route. During the tabu search, all cargo quantities are set to a minimum when evaluating a move. If the move is feasible, the quantities are optimized afterward. A linear programming approach is formulated for the subproblem of quantity optimization, which is solved using a fast heuristic. This heuristic simply maximizes all cargo quantities, and decreases it step wise, starting with the first cargo, until the route becomes feasible. The computational study shows that this approach returns high-quality solutions to real-life cases. Additionally, as compared to column generation approaches, the computational effort is significantly less.

*Vehicle Routing with Metaheuristics* Parallels can be drawn between ship scheduling problems and the well-studied Vehicle Routing Problem with Pickup and Delivery (VRPPD), which is an extension of the classic Vehicle Routing Problem (VRP). The VRPPD is the problem of optimally assigning a fleet of vehicles to satisfy a set of customer demands. These demands are specified by the quantity of cargo that needs to be transported, the pickup location, and the delivery location. Generally, all demands must be satisfied while minimizing operating costs, subject to vehicle capacity constraints. Variations including time windows, also referred to as the Pickup and Delivery Problem with Time Windows (PDPTW), have been extensively studied as well, see [4, 29].

It is well-known that the VRP is a NP-hard problem as it is a generalization of the Traveling Salesman Problem (TSP), which makes finding the optimal solution intractable for large cases [15]. The VRPPD is an extension of the VRP and hence is NP-hard too. Many metaheuristics have been applied to solve the VRP including simulated annealing, deterministic annealing, genetic algorithms, ant systems, tabu search, and neural networks. Tabu search (TS) clearly stands out as one of the best heuristics for the VRP [12].

Fred Glover proposed the tabu search approach back in 1986, with the purpose of enabling Local Search heuristics to overcome local optima. The principle is to allow non-improving moves to escape a local optimum; cycling is prevented

by using tabu lists. This *Meta-Heuristic*, a term also coined by Glover, appeared highly effective and drew the attention of many researchers. Some of the major TS algorithms for the VRP are described in [12]. They also present comparative computational results for these TS algorithms over fourteen benchmark instances. We continue the literature review on tabu search applications to the VRP in Section 4.2.

## 2.4 Summary

Unlike other modes of transport, the literature in maritime transport, especially in the tramp industry, is very scarce. Most research has been devoted to the industrial shipping industry, and despite recent trend toward the tramp industry, literature has not caught up yet.

This chapter presented the little literature in tramp ship scheduling. Furthermore, some papers that addressed the problem of deciding on ship cruising speed are discussed. Also, regarding solution approaches, a link is made with the realm of Vehicle Routing Problems, which are extensively studied in the literature.

To the knowledge of the author, metaheuristics have not yet been applied to tramp scheduling problems in literature. Conversely, the realm of VRPPD has been a testing ground for a wide range of metaheuristics for a long period of time with numerous successes; see [29, Sec 3.2.3] for a survey. It baffles the author that successful applications of computer science methods for the VRPPD has not yet made the transfer to ship routing and scheduling problems. In this thesis, we apply tabu search to solve our tramp scheduling problem. Chapter 4 will continue the discussion around our solution approach. First, mathematical models for our problem are proposed in the next chapter.

# 3. THE TRAMP SHIP SCHEDULING PROBLEM: A MATHEMATICAL MODEL

This chapter presents a formal mathematical model of the Tramp Ship Scheduling with Speed Decisions (TSS-SD). First, we present a general Tramp Ship Scheduling (TSS) model derived from [6, 7] in Section 3.2. However, in [6, 7] they studied and hence modeled flexible cargo sizes. This extension is practical, but is out of the scope of this thesis; we will omit the possibility of flexible cargo sizes. It makes our model unnecessarily more complicated for our research into speed decisions.

Then we extended the general TSS model with speed decisions, which incorporates a model of [31], that relates bunker fuel consumption with sailing speed. This Tramp Ship Scheduling with Speed Decisions (TSS-SD) model is described in 3.3.

We proposed another extension this model by taking soft time windows into account, inspired by [13]. They studied the industrial ship scheduling problem with soft time windows and applied a set partitioning approach. They generated candidate schedules while taking speed decisions into account by deciding on arrival times. We will also make use of soft time windows, as it enables more flexibility in speed decisions. A mathematical model of the Tramp Ship Scheduling with Speed Decisions and Soft Time Windows TSS-SDSTW is presented in Section 3.4. Even though we propose the mathematical model for the TSS-SDSTW, we do not implement them for it is out of the scope of this research; it can be interesting for future research.

Before anything, we present the mathematical notation in Section 3.1. We used, where possible, the same notation as in [6, 7] for consistency.

## *3.1 Notation*

*Sets* We denote the fleet of available ships with $\mathcal{V}$, indexed by $v$. The set of cargoes, and its associated loading ports, $\mathcal{N}_P$ is indexed by $i$, which consist of the set of contracted cargoes $\mathcal{N}_C$ and the set of optional spot cargoes $\mathcal{N}_O$; $\mathcal{N}_P = \mathcal{N}_C \cup \mathcal{N}_O$. Associated with each cargo is a loading port and an unloading port, represented by a node $i$ and $N + i$ respectively, in which $N$ is the total number of cargoes to be serviced during the planning horizon. Note that different nodes may represent the same port. The set of loading nodes is given by $\mathcal{N}_P = \{1, \ldots, N\}$ and the set of unloading nodes by $\mathcal{N}_D = \{N + 1, \ldots, 2N\}$.

Two additional artificial nodes are used for each ship. Let $o(v)$ denote the

origin depot and $d(v)$ the destination depot of ship $v$. The origin depot can both be a port that is last visited or a position at sea at the beginning of the planning horizon. The artificial destination depot denotes the last planned unloading port it visits. So if ship $v$ is unused, $o(v)$ and $d(v)$ represent the same location. The set of all nodes is then given by $\mathcal{N} = \mathcal{N}_P \cup \mathcal{N}_D \bigcup_{v\in\mathcal{V}} o(v) \bigcup_{v\in\mathcal{V}} d(v)$.

A ship may not be feasible to visit some ports, perhaps due to its size. Let $\mathcal{N}_v$ represent the set of feasible ports to ship $v$. Here, $\mathcal{N}_{Pv}$ denotes the set of feasible loading nodes and $\mathcal{N}_{Dv}$ the set of feasible unloading nodes for ship $v$.

Let a network associated with a ship $v$ be given by $(\mathcal{N}_v, \mathcal{A}_v)$ where $\mathcal{A}_v$ represents the set of feasible arcs, which is a subset of $\mathcal{N}_v \times \mathcal{N}_v$. This set is calculated based on time and capacity constraints, and also precedence constraints of visiting a cargo's loading port before visiting its unloading port.

In summary we have the following sets:

| Variable | : Description |
|---|---|
| $\mathcal{V}$ | : The fleet of available ships |
| $\mathcal{N}_C$ | : The set of contracted cargoes and its associated loading node |
| $\mathcal{N}_O$ | : The set of optional cargoes' loading nodes |
| $\mathcal{N}_P$ | : Set of all loading nodes, $\mathcal{P} = \mathcal{N}_C \cup \mathcal{N}_O$ |
| $\mathcal{N}_D$ | : The set of all unloading nodes |
| $\mathcal{N}$ | : The set of all nodes, $\mathcal{N} = \mathcal{N}_P \cup \mathcal{N}_D \bigcup_{v\in\mathcal{V}} o(v) \bigcup_{v\in\mathcal{V}} d(v)$ |
| $\mathcal{N}_v$ | : The set of feasible nodes for ship $v \in \mathcal{V}$, $\mathcal{N}_v \subset \mathcal{N}$ |
| $\mathcal{N}_{P_v}$ | : The set of feasible loading nodes for ship $v \in \mathcal{V}$ |
| $\mathcal{N}_{D_v}$ | : The set of feasible unloading nodes for ship $v \in \mathcal{V}$ |
| $\mathcal{A}$ | : The set of arcs, $\mathcal{A} = \mathcal{N} \times \mathcal{N}$ |
| $\mathcal{A}_v$ | : The set of feasible arcs for ship $v \in \mathcal{V}$, $\mathcal{A}_v \subset \mathcal{A}$ |

*Parameters* The parameters are essentially the configuration of the model. The capacity of a ship $v$ is given by $V_{CAP_v}$. The sailing time of ship $v$ departing from node $i$ and arriving on node $j$ is given by $T_{S_{ijv}}$. The associated costs are denoted by $C_{ijv}$. The time window of a cargo is given by an earliest possible time for start of service $T_{MNi}$ and a latest possible time for start of service $T_{MXi}$ at node $i$. Time required to load or unload cargo at node $i \in \mathcal{N}_P \cup \mathcal{N}_D$ is given by $T_i$. The quantity of cargo $i \in \mathcal{N}_P$ is given by $Q_i$ and its revenue for servicing is given by $R_i$. Let $\pi_i$ denote the profit of servicing a cargo $i \in \mathcal{N}_P$ by a spot charter. Note that $\pi_i$ may also be negative if the revenue created by servicing the cargo is lower than the cost of a spot charter.

For the TSS-SD we require some additional parameters. We need to split up the cost of sailing $C_{ijv}$ in fixed and variable costs. For sailing each arc $v \in \mathcal{V}, (i,j) \in \mathcal{A}_v$ a ship needs to pay fixed port costs, denoted by $P_{ijv}$. Additionally, variable fuel costs are denoted by $F_{ijv}$, which depend on the cruising speed. Here, the value of $F_{ijv}$ is set to the fuel costs associated with a nominal cruising speed $V_{0_v}$ of ship $v \in \mathcal{V}$ sailing on arc $(i,j) \in \mathcal{A}_v$.

When we consider soft time windows in the TSS-SDSTW model, we require parameters that represent the outer time windows, i.e. $[E_i, L_i]$. These values are derived from the given time windows $[T_{MNi}, T_{MXi}]$ using parameter $D_i^{MAX}$ which denotes the maximum time window violation. Also, we introduce a penalty parameter $\rho_i$ to penalize out-of-time-window services.

Summarized we have the following parameters:

| Variable | : Description |
|---|---|
| $V_{CAP_v}$ | : The capacity of ship $v \in \mathcal{V}$ |
| $T_{S_{ijv}}$ | : Sailing time of ship $v \in \mathcal{V}$ from node $i \in \mathcal{N}_v$ to node $j \in \mathcal{N}_v$ |
| $C_{ijv}$ | : Cost of sailing ship $v \in \mathcal{V}$ from node $i \in \mathcal{N}_v$ to node $j \in \mathcal{N}_v$ |
| $T_{MNi}$ | : Earliest possible time for start of service at node $i \in \mathcal{N}_P \cup \mathcal{N}_D$ |
| $T_{MXi}$ | : Latest possible time for start of service at node $i \in \mathcal{N}_P \cup \mathcal{N}_D$ |
| $T_i$ | : Time to load or unload cargo at node $i \in \mathcal{N}_P \cup \mathcal{N}_D$ |
| $Q_i$ | : Quantity of cargo $i \in \mathcal{N}_P$ |
| $R_i$ | : Revenue of servicing cargo $i \in \mathcal{N}_P$ |
| $\pi_i$ | : Profit of servicing cargo $i \in \mathcal{N}_P$ by a spot charter |
| $P_{ijv}$ | : Fixed port cost for ship $v \in \mathcal{V}$ on arc $(i,j) \in \mathcal{A}$ |
| $F_{ijv}$ | : Fuel cost for ship $v \in \mathcal{V}$ to sail on arc $(i,j) \in \mathcal{A}$ at nominal speed |
| $V_{0_v}$ | : Nominal speed of ship $v \in \mathcal{V}$ |
| $D_i^{MAX}$ | : Maximum time window violation for cargo $i \in \mathcal{N}_P \cup \mathcal{N}_D$ |
| $E_i$ | : Outer time window, where $E_i = T_{MNi} - D_i^{MAX}$ |
| $L_i$ | : Outer time window, where $L_i = T_{MXi} + D_i^{MAX}$ |
| $\rho_i$ | : Penalty parameter for out-of-time-window services of cargo $i$ |

*Decision variables* The decision variables are the variables that we alter in order to optimize our objective function, which is in this case to maximize the profits. We have a ship routing and scheduling problem, hence the decisions are to make optimal schedules for the ships available. We use the binary flow variable $x_{ijv} \in \{0,1\}, v \in \mathcal{V}, (i,j) \in \mathcal{A}_v$ to represent the decision to let ship $v$ sail directly from node $i$ to node $j$. Another decision variable that we introduce is $s_i \in \{0,1\}$, which decides on deploying a spot charter for cargo $i \in \mathcal{N}_P$.

For the TSS-SD we have an additional decision variable for the sailing speed. Let us introduce $\nu_{ijv}$, which is a continuous nonnegative decision variable for the sailing speed of ship $v \in \mathcal{V}$ on arc $(i,j) \in \mathcal{A}_v$.

In summary we have the following decision variables:

| Variable | : Description |
|---|---|
| $x_{ijv}$ | : Binary variable of letting ship $v \in \mathcal{V}$ sail on arc $(i,j) \in \mathcal{A}_v$ |
| $s_i$ | : Binary variable of serving cargo $i \in \mathcal{N}_P$ with a spot charter |
| $\nu_{ijv}$ | : Cruising speed of ship $v \in \mathcal{V}$ on arc $(i,j) \in \mathcal{A}_v$ |

*Model variables* Furthermore, we introduce the following model variables that are either derived from parameters, decision variables, or both.

The time variable $t_{iv}, v \in \mathcal{V}, i \in \mathcal{N}_V$ denotes the time of start of service at node $i$ with ship $v$. This variable is derived from the present schedule of ship $v$ and the sailing times between the scheduled nodes. For nodes $i$ that are not visited by the respective ship $v$, the value of $t_{iv}$ is 0. Let $l_{iv}, v \in \mathcal{V}, i \in \mathcal{N}_V$ represent the total load onboard ship $v$ right after departure of node $i$.

The model variables are summarized below:

| Variable | : Description |
|---|---|
| $t_{iv}$ | : The time of start of service at node $i \in \mathcal{N}_V$ with ship $v \in \mathcal{V}$ |
| $l_{iv}$ | : Total load on ship $v \in \mathcal{V}$ after visiting node $i \in \mathcal{N}_V$ |

## 3.2 General tramp ship scheduling model

This section presents a mathematical graph representation of the general Tramp Ship Scheduling (TSS) problem. It is allowed to carry multiple cargoes on one ship, despite what load is already onboard, provided that there is enough capacity. The cargoes have hard time windows for both pickup and delivery. All contracted cargoes required to be shipped, albeit with the help of a spot charter. The objective is to maximize profits by optimally scheduling the available fleet, and the use of spot charters. Note that with the help of spot charters, a feasible solution always exists. The general TSS model is an adjusted model of [6, 7]; we do not incorporate flexible cargo quantities and allow spot charters to be deployed on spot cargoes as well:

$$\max \left[ \sum_{i \in \mathcal{N}_P} \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_v} R_i x_{ijv} - \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}_v} C_{ijv} x_{ijv} + \sum_{i \in \mathcal{N}_P} \pi_i s_i \right], \tag{3.1}$$

subject to

$$\sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_v} x_{ijv} + s_i = 1, \qquad \forall i \in \mathcal{N}_C, \tag{3.2}$$

$$\sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_v} x_{ijv} + s_i \leq 1, \qquad \forall i \in \mathcal{N}_O, \tag{3.3}$$

$$\sum_{j \in \mathcal{N}_{P_v} \cup d(v)} x_{o(v)jv} = 1, \qquad \forall v \in \mathcal{V}, \tag{3.4}$$

$$\sum_{i \in \mathcal{N}_v} x_{ijv} - \sum_{i \in \mathcal{N}_v} x_{jiv} = 0, \qquad \forall v \in \mathcal{V}, j \in \mathcal{N}_v \backslash \{o(v), d(v)\}, \tag{3.5}$$

$$\sum_{i \in \mathcal{N}_{D_v} \cup o(v)} x_{id(v)v} = 1, \qquad \forall v \in \mathcal{V}, \tag{3.6}$$

$$x_{ijv}(t_{iv} + T_i + T_{S_{ijv}} - t_{jv}) \leq 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \tag{3.7}$$

$$T_{MNi} \leq t_{iv} \leq T_{MXi}, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_v, \tag{3.8}$$

$$x_{ijv}(l_{iv} + Q_j - l_{jv}) = 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v | j \in \mathcal{N}_{P_v}, \tag{3.9}$$

$$x_{i,N+j,v}(l_{iv} - Q_j - l_{N+j,v}) = 0, \qquad \forall v \in \mathcal{V}, (i,N+j) \in \mathcal{A}_v | j \in \mathcal{N}_{P_v}, \tag{3.10}$$

$$0 \leq l_{iv} \leq V_{CAP_v}, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{P_v}, \tag{3.11}$$

$$t_{iv} + T_i + T_{S_{i,N+i,v}} - t_{N+i,v} \leq 0, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{P_v}, \tag{3.12}$$

$$\sum_{j \in \mathcal{N}_v} x_{ijv} - \sum_{j \in \mathcal{N}_v} x_{j,N+i,v} = 0, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{P_v}, \tag{3.13}$$

$$x_{ijv} \in \{0,1\}, \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \tag{3.14}$$

$$s_i \in \{0,1\}, \qquad \forall i \in \mathcal{N}_C. \tag{3.15}$$

The objective is to maximize the total profit during the planning horizon, given by the objective function in (3.1). Here, the first term accounts for the total revenue generated by transporting cargoes, be it contracted or optional. The second term

calculates the variable operating costs, i.e. total sailing and port costs of the entire fleet. Note that the fixed operating costs are neglected as they do not influence the model's outcome.Term three calculates the profits (positive or negative) of using spot charters to transport cargoes.

Constraints (3.2) make sure that all contracted cargoes are served, either with an own ship or with a spot charter. Constraints (3.3) state that all optional cargoes are not served more than once.

Constraints (3.4)-(3.6) formulate the sailing flow of a ship, and ensures that it departs from an artificial origin node and arrives at the artificial destination node.

Calculations regarding service times are done in constraints (3.7). Here, the time of starting a service at node $j$ cannot be earlier than the start of service at node $i$ plus the service duration and the sailing time between the nodes. Note that the inequality sign allows for waiting at node $j$, if the ship arrives before the earliest allowable time of service $T_{MNj}$. Constraints (3.8) make sure that loading of unloading is done within the specified time window.

Load of the ships are calculated in constraints (3.9) and (3.10) after loading and unloading respectively. These loads are required to stay within a ship's capacity at all times, which is stated by constraints (3.11).

Ships that visit a loading node $i \in \mathcal{N}_P$ are also required to deliver the cargo at its unloading node $N + i \in \mathcal{N}_D$. This is captured by constraints (3.13). Furthermore, an unloading node cannot be visited before its loading node is visited, which is stated by constraints (3.12).

Additionally, we require the decision variables $x_{ijv}$ and $s_i$ to be binary in constraints (3.14)-(3.15).

Before going into solving this model, we first present an extension of this general TSS model that incorporates cruising speed decisions, followed by another extension that applies soft time windows.

## 3.3 Tramp ship scheduling model with speed decisions

A ship can reduce its bunker fuel consumption, hence its operating costs, significantly by sailing slower than nominal. According to [27], the bunker fuel consumption of the main engines of a motor ship is directly related to the third power of the speed. This means that reducing cruising speed by 20%, it reduces bunker fuel consumption approximately by 50%. By sailing at half speed, this will yield a staggering 80% saving in fuel consumption. The relation is given by (3.16); even though this equation is an empirical one, it remains a useful approximation.

$$F = \left(\frac{V}{V_0}\right)^3 F_0, \tag{3.16}$$

where $V$ is the sailing speed, $V_0$ is the nominal speed, $F_0$ is the nominal fuel consumption and $F$ is the actual fuel consumption. Note that not all speeds from zero to nominal are technically feasible for a motor ship; a minimum speed of about 50% is required for proper steerage. This model has been applied in [31], who studied the tradeoff between slow steaming and saving fuel costs on one hand, and the opportunity costs of delaying arrival on the other hand. A medium sized ship that burns 40 tons of bunker fuel a day may save 20 tons by reducing sailing speed with 20%. With the prices for a metric ton of bunker fuel ranging from $400 to $500 (July 2009), this may

result up to savings of $10,000 per day per ship. Fuel consumption is the main driver of operating costs of a ship.

We will make use of (3.16) to calculate operating costs in the objective function of our TSS-SD model:

$$\max \left[ \sum_{i \in \mathcal{N}_P} \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_v} R_i x_{ijv} - \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}_v} x_{ijv} \left( \left( \frac{\nu_{ijv}}{V_{0_v}} \right)^3 F_{ijv} + P_{ijv} \right) + \sum_{i \in \mathcal{N}_P} \pi_i s_i \right]. \tag{3.17}$$

In (3.17) we adjusted the second term of (3.1) by splitting the sailing cost parameter $C_{ijv}$ into fixed and variable costs. $P_{ijv}$ are fixed port costs that need to be payed when arc $(i,j) \in \mathcal{A}_v$ is sailed by ship $v \in \mathcal{V}$. The variable bunker fuel costs are denoted by $F_{ijv}$ and is dependant on the sailing speed. This introduces a new decision variable for sailing speed $\nu_{ijv}, (i,j) \in \mathcal{A}_v, v \in \mathcal{V}$, which is continuous.

Sailing slower obviously result in a longer travel time and a later arrival time at the next destination. When sailing twice as slow, the sailing time will be twice as long. We need to adjust constraints (3.7) and (3.12):

$$x_{ijv}(t_{iv} + T_i + T_{S_{ijv}} \left( \frac{V_{0_v}}{\nu_{ijv}} \right) - t_{jv}) \leq 0, \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v, \tag{3.18}$$

$$t_{iv} + T_i + T_{S_{i,N+i,v}} \left( \frac{V_{0_v}}{\nu_{ijv}} \right) - t_{N+i,v} \leq 0, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{P_v}. \tag{3.19}$$

Additionally, we need to impose a speed range constraint for the new continuous decision variable $\nu_{ijv}$. Furthermore, we require $\nu_{ijv}$ to be nonnegative, but this is already implied by constraints (3.20).

$$\frac{1}{2} V_{0_v} \leq \nu_{ijv} \leq V_{0_v}, \qquad \forall v \in \mathcal{V}, (i,j) \in \mathcal{A}_v. \tag{3.20}$$

The TSS-SD can now be formulated by maximizing (3.17) subject to (3.2)-(3.6), (3.8)-(3.11), (3.13)-(3.15) and (3.18)-(3.20).

## 3.4 Tramp ship scheduling model with speed decisions and soft time windows

We propose another mathematical model below for future research. We suggest further extending the TSS-SD model by allowing for soft time windows, which widens the range of possible speed decisions. Soft time windows give rise to the opportunity to deliver some cargoes slightly later against a penalty, and in return a higher saving in fuel costs can be achieved and perhaps an even better overall schedule.

With each cargo is associated an *inner time window* and an *outer time window*, which is inspired by [13]. They applied soft time windows on both loading and unloading nodes, and allowed for both early and late services. However, considering a pickup and delivery problem, we argue that loading a cargo earlier is infeasible, as the cargo has not arrived yet. We do not allow for early pickup in our TSS-SDSTW model.

The inner time window for each cargo $i \in \mathcal{N}_P \cup \mathcal{N}_D$ is given by $[T_{MNi}, T_{MXi}]$. Note that this time window is for both loading and unloading nodes. Further, let $D_i^{MAX}$ represent the maximum violation of the inner time window for a given node

$i \in \mathcal{N}_P \cup \mathcal{N}_D$. Now, the outer time windows $[E_i, L_i]$ are derived from the inner time windows using $D_i^{MAX}$, where $E_i = T_{MNi} - D_i^{MAX}$ and $L_i = T_{MXi} + D_i^{MAX}$. Note here that for loading nodes, the outer time windows are given by $[T_{MNi}, L_i]$, as we do not allow for early pickup. All cargoes must be within the outer time windows, but preferably within the inner time windows, as we penalize tardy pickups and deliveries. In [13] different penalty functions are examined, including a linear function, a quadratic function, and a constant penalty. We use a linear penalty function for simplicity; the number of days outside the inner time window, but within the outer time window, is multiplied with a penalty factor $\rho_i$. The value of this parameter is negotiated between the clients and the shipping company in a contract, and generally depends on the value of the cargo to be shipped. This penalty factor can also be used to prioritize certain cargoes.

For this model, we need to change the time window constraints in (3.8). We replace them with the following sets of constraints:

$$E_i \leq t_{iv} \leq L_i, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{D_v}, \tag{3.21}$$

$$T_{MNi} \leq t_{iv} \leq L_i, \qquad \forall v \in \mathcal{V}, i \in \mathcal{N}_{P_v}. \tag{3.22}$$

Also, we need to incorporate a penalty in the objective function, which becomes:

$$\max \Bigg[ \sum_{i \in \mathcal{N}_P} \sum_{v \in \mathcal{V}} \sum_{j \in \mathcal{N}_v} R_i x_{ijv} - \sum_{v \in \mathcal{V}} \sum_{(i,j) \in \mathcal{A}_v} x_{ijv} \left( \left( \frac{\nu_{ijv}}{V_{0_v}} \right)^3 F_{ijv} + P_{ijv} \right) + \sum_{i \in \mathcal{N}_P} \pi_i s_i - \sum_{v \in \mathcal{V}} \sum_{i \in \mathcal{N}_v} \Phi(t_{iv}) \Bigg], \tag{3.23}$$

where $\Phi(t_{iv})$ is a penalty function that calculates the penalty for time window violations, which is defined in (3.24).

$$\Phi(t_{iv}) = \begin{cases} (T_{MNi} - t_{iv})\rho_i, & \text{if } E_i \leq t_{iv} \leq T_{MNi}, \\ (t_{iv} - T_{MXi})\rho_i, & \text{if } T_{MXi} \leq t_{iv} \leq L_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.24}$$

Now we can formulate the TSS-SDSTW model by maximizing (3.23) subject to (3.2)-(3.6), (3.9)-(3.11), (3.13)-(3.15), (3.18)-(3.20) and (3.21)-(3.22).

## 3.5 Summary

We formulated three mathematical models in increasing complexity. First, we present a general Tramp Ship Scheduling (TSS) model, which is then expanded into a Tramp Ship Scheduling with Speed Decisions (TSS-SD) model. Finally, we proposed another model which incorporates soft time windows, resulting in a third model, the Tramp Ship Scheduling with Speed Decisions and Soft Time Windows (TSS-SDSTW). This latter model will be out of the scope of this research, but we propose it here anyway for future research.

We compare the performance of the first two models with respect to the maximum profits achieved, and computational effort. Including speed decisions into the general TSS model increases the solution space and model complexity. Most likely,

this will yield a better optimal solution, but as a consequence, it would require more computational time. The quantitative comparison is presented in Chapter 6.

First, we need to devise an efficient algorithm to solve the models. This is described in the following chapters.

# 4. TABU SEARCH

Along with the rise of complexity theory in the 1970s came the realization that most problems in operations research are NP-hard, see for example [15]. This implied that the search for efficient algorithms to find the global optimum had to be abandoned. New heuristics that provided quick and sub-optimal solutions were proposed under the name *metaheuristics*. Metaheuristics is a term originally coined by Glover in 1986, to describe solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space [20]. The literature on metaheuristics has increased tremendously during the last decades, catalyzed by ever increasing computational power.

Parallels can be drawn between our ship scheduling problem and the well-studied Vehicle Routing Problem with Pickup and Delivery (VRPPD), which is an extension of the classic Vehicle Routing Problem (VRP). The VRPPD is the problem of optimally assigning a fleet of vehicles to satisfy a set of customer demands. These demands are specified by the quantity of cargo that needs to be transported, the pickup location, and the delivery location. Variations including time-windows have been considered too. These problems have been studied for over 30 years, and a survey can be found in [4, sec. 4.2].

Despite the extensive amount of literature describing numerous different applications of metaheuristics for the VRP(PD), papers on ship scheduling with metaheuristics remain rare, regardless of the (practical) relevance.

Numerous different solution approaches to the VRP have been examined in the literature. It is well-known that the VRP is a NP-hard problem, which makes finding the optimal solution intractable for large cases. The VRPPD is a generalization of the VRP and hence is NP-hard too. Many metaheuristics have been applied to solve the VRP including simulated annealing, deterministic annealing, genetic algorithms, ant systems, tabu search, and neural networks. Tabu search clearly stands out as one of the best heuristics for the VRP [12]. It is a highly efficient solution approach that gained enormous popularity in the field of combinatorial optimization for its performance.

For our ship scheduling problem, which is formulated in the previous chapter using a graph representation, it is fairly intuitive to understand the search process of tabu search. As opposed to some other metaheuristical approaches, e.g. genetic algorithm or neural networks, the tabu search is not a black box approach, in that the search can understandably be followed and traced back as it is a deterministic algorithm. Furthermore, useful knowledge of the problem at hand can be easily incorporated, for tabu search is very customizable, for example by defining logical neighborhood structures. Another reason why we chose for tabu search, is the speed of the algorithm. In very short time, a very good solution can be obtained, which is practical for our dynamic tramp ship scheduling problem. In practice, when spot orders arrive at the respective shipping company, it needs to decide timely whether they accept the order

or neglect it; speed is a crucial factor.

This chapter first gives a brief introduction to the general aspects of tabu search, along with a brief discussion of a few papers that applied tabu search to the VRP(PD). Then we describe our tabu search implementation, tailored for our tramp ship scheduling problems.

## 4.1 Introduction to Tabu Search

This section provides a brief general introduction into tabu search, compiled from [21, 16, 12]. Tabu search is a term originally coined by [19]. In short, tabu search can be described as a local search technique, that allows for non-improving moves to overcome local optima. Local search techniques are iterative search procedures, that improves on a feasible initial solution by considering local modifications, one step at a time. It hovers the search space until a solution that is deemed optimal is found. Whereas local search techniques always cope with the problem of getting stuck in local optima, tabu search allows for non-improving moves to escape from these.

Tabu search explores the solution space by moving from a solution $x_t$ at iteration $t$ to the best solution $x_{t+1}$ in a subset of the neighborhood $N(x_t)$, until a certain stopping condition. Here, $x_{t+1}$ is not necessarily improving upon $x_t$, hence the search might go back to old solutions and keep cycling over a sequence of solutions. This is prevented using tabu lists; attributes of recent moves are recorded and potential solutions that contain these attributes are not allowed for a number of iterations. This is often referred to as short term memory.

Whereas in some applications it suffices to use short term memory to produce quality solutions, generally tabu search becomes significantly stronger by including longer term memory and its associated strategies. Two common strategies of using long term memory are diversification and intensification. Diversification enables a wider exploration of the solution space, for example by forcing the search into unexplored areas of the solution space. Conversely, intensification performs an intensified search around the presently encountered elite solutions.

Many different tabu search implementations are possible, as tabu search allows the user to tailor the implementation to a specific problem. Below, we give an introduction to the several aspects of tabu search, where we remain our focus on applications of tabu search to the VRP.

### 4.1.1 Neighborhood structure

The most important aspect of designing a tabu search heuristic, is its neighborhood structure denoted by $N(x_t)$. This defines the area of the search space being considered at any iteration, i.e. solutions that can be reached with a single local transformation or move. Essentially, they describe the possible moves that may be considered during the search. These neighborhood structures must be carefully defined for the problem at hand; one must make best use of the understanding and knowledge one has regarding the problem at hand.

Let us illustrate this by giving an example of a simple neighborhood structure for the VRP. At each iteration, we consider selecting a node from the current route, and inserting it in another route or in the same route with another position. An important feature is the insertion heuristic, which defines the way of inserting a node in a target route. This could be done using random insertion or the node could be inserted in the

best possible way. Obviously, numerous different insertion heuristics can be thought of, just as numerous neighborhood structures are possible.

Two main ways to define a neighborhood structures for the VRP are $\lambda$-interchanges and *ejection chains*. $\lambda$-interchange allows for exchanging up to $\lambda$ nodes between two routes. Usually, $\lambda$ is restricted to 1 or 2 to limit the number of possible moves to be considered. Several variants can be thought of, such as excluding some nodes from interchanging, or local reoptimization of the routes in which interchanges have taken place. Ejection chains first identifies a number of routes in which nodes are moved around from route $k_1$ to $k_2$, from $k_2$ to $k_3$, etcetera. Note that the routes that are identified are not necessarily different routes, which allows for intra-routes moves as well.

When designing the neighborhood structure, one must consider the trade-off between the quality of the search and the computational effort. Complicated neighborhood structures that search a wide area of the solution space will likely come up with better quality solutions, but the computational effort that is required to perform such search may be excessive, making the approach impractical. Therefore, we need to strike a balance and come up with simple, yet efficient neighborhood structures that only considers a select but promising moves.

One way to achieve computational efficiency is to apply *probabilistic ts*. As opposed to evaluating every possible solution in the neighborhood $N(x_t)$, which may be computationally expensive, one could consider only a random sample $N'(x_t) \in N(x_t)$. This added randomness also adds to diversifying the search, which helps to prevent cycling. This also implies that less lengthy tabu lists suffice to already broadly explore the search space, since sub-optimal moves will be undertaken. A major drawback of probabilistic tabu search is that good moves may not be sampled in $N(x_t)$, hence best solutions returned may not be a local optimum. Applying a local improvement phase on the best found solution at the end of the tabu search is a good way to overcome this problem. Alternatively, *intensification* can be used to further scrutinize the neighborhood of the presently found best solution, which will be discussed below in Section 4.1.3.

Another way to reduce the size of the neighborhood can be achieved through *candidate lists*. These are lists of candidate neighborhood moves that will be considered for the next move, and are generated according to their potential. One may for example use a *surrogate objective* that is less costly to evaluate, as compared to the real objective, but still is a good indicator whether a move has potential at all. From this subset of candidate moves, all true objectives are then evaluated and the best move is chosen.

Another issue is whether or not we allow for infeasible solutions during the search. By doing so, we might reach better feasible solutions in two iterations, through an infeasible solution, that otherwise would be out of reach from the present solution when considering only feasible moves. When a problem has many constraints, accounting for every constraint may strongly restrict the search process. This is for example the case for our TSS-SDSTW, where we have numerous constraints that restrict many potential good moves. For these problems, *constraint relaxation* might be a good solution. Using simple neighborhood structures, a large search space can still be achieved. Constraint violations can then be penalized in the objective function. By choosing to do so, also requires one to choose the according penalty weights. One could make use of *self-adjusting penalties*, which are weights that adjust dynamically during the search. Weights are decreased (or increased for the maximization problem) by a factor, when only feasible solutions are encountered in the past few iterations; when only infeasible

solutions were encountered in the most recent search history, the penalty weights are increased (or decreased). These adjustments may also be made systematically, known as *strategic oscillation*, in order to drive the search to other areas of the search space, i.e. diversification (see Section 4.1.3.)

### 4.1.2 Short term memory

One of the distinctive characteristics of tabu search is that the neighborhood structure for each solution alters as the search progresses. Primarily, this is achieved through the use of a *tabu list*, stored in a *short term memory* of the search. This list, usually limited, records potential moves that are declared *tabu* to prevent the search from cycling when non-improving moves are taken. For instance, when a node has been moved from one route to the other, the move of moving the same node back again can be declared tabu for a number of iterations. This number of iterations may be fixed, random or dynamic as the search progresses, and is called the *tabu tenure*. Tabus can also direct the search toward some unexplored areas of the search space, as otherwise not considered moves will be taken.

As an example, consider the following tabu for the VRP, where a node $v_1$ has just been moved from route $R_1$ to route $R_2$. Now we may declare the move of node $v_1$ from $R_2$ back to $R_1$ tabu by for example record it as the triplet $(v_1, R_2, R_1)$. Note that this tabu does not prohibit the return of $v_1$ back to $R_1$ if it first goes to $R_3$. Alternatively, we could impose another tabu that constrains $v_1$ from returning to $R_1$ at all, by recording $(v_1, R_1)$. An even stronger tabu would be to not consider moving $v_1$ to any route at all, which would be recorded as simply as $(v_1)$.

Generally, tabu lists are implemented as circular lists of fixed length. It has been shown that fixed-length tabu lists can not guarantee to prevent cycling. Alternatively, one could consider varying the length of the list during the search, or generate the tabu tenure of each move randomly within some interval. The latter approach requires a different recording mechanism, by using tags in an array, which would depict the iteration number until which a move may be considered again.

Sometimes a tabu list may be too powerful and restrict the search from exploring the search space effectively. Attractive moves may be on the tabu list, even though there is no danger of cycling. It is therefore useful to deploy heuristical tricks to delete some entries in the tabu list when appropriate. These mechanisms are referred to as *aspiration criteria*. The most commonly applied aspiration criteria is to allow a tabu move, if it results in a better solution than the current best-known solution. Other more complicated aspiration criteria can be thought of, but are generally rare.

### 4.1.3 Long term memory

The inevitable problem of Local Search techniques, in fact all of metaheuristics, is that the search may be too local. Most time of the search is spent in only a small region of the search space, perhaps very distant to the optimal solutions. *Diversification* is a term that refers to algorithmic techniques that prevents a search to be too local, and forces the search into unexplored areas of the search space. These techniques generally make use of *long-term memory*, which can be used to assess how the search is progressing. For instance, a *frequency memory* records the total number of iterations that a certain solution component has been present in the current solutions, or that have involved in certain moves. For example, one could record how many times each node has moved from its current route.

Two of the most common diversification techniques are *restart diversification* and *continuous diversification*. Restart diversification stops the present search, and restarts the search from a starting point that has not thoroughly been explored yet. For example, nodes that have not been moved yet will be forced into other routes. The search will resume in this new area of the search space. Another way of inducing diversification, *continuous diversification*, enables diversification by biasing the evaluation of the regular search progress. Additional terms in the (surrogate) objective may be introduced, to penalize frequent solution components, inviting the search algorithm to try out different moves.

Another class of techniques that make use of a long term memory, is *intensification*. These techniques are opposite from diversification techniques, and explores more thoroughly certain promising regions. When a good solution is found through a general broad search, an even better solution might be found by focussing the search in the local neighborhood of this good solution. This could for example be achieved by applying a more complex neighborhood structure, such as ejection chains or more complex insertion moves, around some of the best known solutions.

## 4.2 Tabu Search Algorithms for the Vehicle Routing Problem

In the previous section we gave a brief introduction in the general ideas of tabu search. This section presents some distinctive features of the most successful implementations of tabu search to the VRP, as described by [12, 24]. It gives the reader an idea of the possibilities and flexibilities in tailoring a tabu search algorithm.

### 4.2.1 Taburoute

The neighborhood structure of the *Taburoute* algorithm [18] is defined by all solutions that can be reached by removing a node from its current route, into another route that contains one of its closest neighbor nodes. The insertion heuristic that is applied is described in [17], called a Generalized Insertion (GENI) procedure. To limit the neighborhood size, only a few randomly selected subset of nodes will considered for moving.

Moreover, the search procedure allows for infeasible solutions, and contains penalty terms in the objective function to account for over capacity and over duration. The weights of the penalty terms are self-adjusting parameters, that double in value when 10 consecutive solutions were infeasible, and get divided by two when the previous 10 solutions were all feasible. This procedure diversifies when the search is local, and intensifies when the search is too divers. Occasionally, Taburoute applies a post-optimization routine after inserting a node, using the Unstringing and Stringing (US) heuristic for the TSP, also described in [17].

Regarding short term memory, Taburoute prevents any node that has been removed from a certain route, to be reinserted for a number of iterations, randomly drawn between 5 and 10. This implies that no tabu lists are used, but random tabu tags. Also, nodes that have been moved around frequently are penalized according to their absolute frequency, to incentivize moving around other less frequently moved nodes, to achieve more diversification.

Two intensification techniques are deployed by Taburoute. Firstly, before the main tabu search starts, a limited search is performed on several generated initial solutions, after which the best solution is chosen for the real search. This is called a *false start*. Secondly, during the main search, when after a number of iterations no improving

moves have been made, a more intense search is performed by using a more complex neighborhood structure.

Let $n$ depict the number of nodes, and $m$ the number of vehicles. Further, let $q$ denote the size of the randomly selected subset of nodes for consideration. The stopping criteria is denoted by $k$, which is the number of consecutive iterations that the search may go on without improvement. Taburoute can now be summarized briefly by the following three search phases.

First phase is the initialization phase where the false start technique is applied. Generate $\lceil \sqrt{n}/2 \rceil$ initial solutions and perform search with $q = 5m$ and $k = 50$.

Second phase is where the main tabu search takes place. Here we take the best solution found in the first phase as the initial solution for the main search, which will be performed with $q = 5m$ and $k = 50n$.

The final phase is the intensification phase. The best solution of the second phase is taken as the current solution, and an intensified search is performed. Half of the most often moved nodes in the first two phases are now selected for further scrutinization, with $k = 50$.

### 4.2.2 Taillard's Algorithm

*Taillard's algorithm*, developed in 1993, one year earlier than Taburoute, applies a very simple neighborhood structure, i.e. the 1-interchange generation mechanism. This makes the search computational very efficient, enabling more iterations in the same computation time, with respect to Taburoute. No infeasible solutions are allowed during the search.

Tabu mechanisms are the same as in Taburoute, or better, Taburoute uses the same tabu mechanisms as in Taillard's algorithm. Random tags are attached to moves indicating the number of iterations that the move is not allowed to be reversed.

The main feature of Taillard's algorithm is the decomposition scheme, that allows for parallel computing. The main problem is split into smaller subregions, which represent subproblems that may be distributed on different processors. Periodically, the boundaries of the subregions are altered to impose diversification.

A computational study in [18] shows that Taburoute outperforms the best existing heuristics on fourteen CMT benchmark data [11]. This computational study was performed in 1994, and also included the Taillard's algorithm, which was published one year earlier. Whereas [18] claims to outperform Taillard's algorithm, the computational comparison studies on the same benchmark data in [12] and [24], show that Taillard's algorithm clearly dominates Taburoute. It furthermore states that Taillard's algorithm remains one of the best tabu search heuristics, as it produced twelve best known solutions out of the fourteen CMT instances.

### 4.2.3 Adaptive Memory Procedure

The concept of *Adaptive Memory* is developed by [30], is a probabilistic technique to intensify and diversify the search. The technique can very well be applied to many heuristics as an additional performance improver. It has, for instance, improved two solutions of the Taillard's algorithm on the CMT benchmark data.

A pool of good solutions is maintained in the Adaptive Memory, which are dynamically updated throughout the search. From time to time, some of these solutions are chosen for recombination; some elements are extracted from several solutions, and

recombined to produce a new good solution. Here, routes that belong to the best solutions carry a larger weight, hence a higher probability of extraction [1]. Recombining routes of different solutions may result in multiple occurrences of certain nodes. This must be avoided during the selection process, where only partial solutions are selected, that will be fixed afterwards using a construction heuristic.

### 4.2.4 Granular Tabu Search

The *Granular tabu search* developed by [34] is a very efficient algorithm that produces excellent results, with only a fraction of the computational effort. Since practical problems become increasingly more complex and instances grow in size, and fast sub-optimal solutions are pragmatic, the Granular tabu search appears to be a very promising concept.

Here, the main idea is that the granular neighborhoods considered during the search is drastically restricted, by permanently removing long edges that are unlikely to contribute to the optimal solution. The Granular tabu search successfully applies a candidate-list strategy, downsizing the edges to be considered to a mere 10 to 20 percent of all edges.

Edges to be considered are required to be shorter than a certain threshold. This threshold is derived from a *sparsification parameter* provided by the user, generally in the interval $[1.0, 2.0]$, and the average edge cost of a good feasible solution obtained by a fast heuristic, e.g. by means of the Clarke and Wright algorithm. The restricted edge set is given by $E(\nu) = \{(v_i, v_j) \in E : c_{ij} \leq \nu\} \cup I$, where $I$ is a set of important edges that will be considered anyhow, e.g. those close to the depot, and where $\nu$ depicts the threshold value.

During the search, the number of edges considered in the candidate-lists are adjusted dynamically through the sparsification factor. When diversification is required, the threshold will increase, allowing more edges to be considered by the tabu search. After a number of iterations, the threshold will be reset to the original value, and the search continues.

### 4.2.5 Reactive Tabu Search

A reactive tabu search algorithm enhances the classical tabu search by automatically adjusting search parameters based on the state and quality of the search. The essence of the algorithm is to reduce the tenure value when the search progresses well, and increase tenure when the search seems to be stuck. One could think of adjusting the tenure value every iteration, based on the previous move. The reactive tabu search algorithm implemented by [28] increases the tenure value by a factor 1.2 when a solution is revisited in a cycle length of 50 iterations. Conversely, the tenure is multiplied by a factor 0.9 when a solution is visited twice within 50 iterations.

[28] applied a reactive tabu search for the PDPTW. They also defined explicit neighborhood structures that take precedence and coupling constraints into account, to account for the PDPTW problem structure. A hierarchical search methodology is used to alternate between different neighborhoods to explore different solution spaces. They compare their algorithm with other state-of-the-art heuristics for the VRP, and achieve superior performance on a benchmark data for the VRPPD.

[1] This may resemble Evolutionary Algorithms, where the search progresses only through the recombination of a pool of solutions, and selection is made according to the solution's fitness.

## 4.3 Summary

Tabu search is a popular and effective local search meta-heuristic for NP-hard problems. This chapter presented a general introduction on the topic of tabu search and its several main aspects, and touched on some of its applications in the literature of Vehicle Routing Problems. Examples of implementations are briefly described to illustrate the flexibility and possibility of different aspects of tabu search.

Even though the VRP, and our TSS problem do not differ that much, it still is a significant different problem. As a result, the tabu search algorithm needs to be customized in order to search effectively. Inspired by the extensive literature on tabu search applications to the VRP, we developed our own tabu search algorithm, tailored for our Tramp Ship Scheduling problem.

# 5. TRAMP TABU SEARCH

In the previous chapter, we gave a brief introduction to tabu search with its several aspects, and showed some examples of implementations in the literature of Vehicle Routing. This section describes our proposed implementation of a tabu search algorithm for the Tramp Ship Scheduling (TSS) problem, we call it the Tramp Tabu Search (TTS). For our implementation in Java, we made use of the open source tabu search framework OpenTS version 1.0 by COINOR[1].

The TSS differs with the PDPTW on several grounds, including:

- The ships in the TSS do not all start and end at a depot
- Available fleet is a given, with their locations
- Orders in the TSS may be chartered out
- Optional cargoes in the TSS may be neglected if not profitable
- Objective is to maximize profits in the TSS, not to minimize total distance

Despite the differences between the TSS and the PDPTW, it remains useful to look at their similarities. To this end, we developed our tabu search algorithm first for a PDPTW, and verified its performance on PDPTW benchmark data of [26]. The computational results are presented in Section 6.1.

After we verified our algorithm, and tuned it to match state-of-the-art performance, we extended the model to solve for our TSS, TSS-SD and TSS-SDSTW models subsequently.

## *5.1 Initial feasible solution*

Before we can start our TTS, we require an initial feasible solution. We can generate an initial feasible solution using a simple greedy heuristic, which is to insert the orders one by one, and assign them to the best ship available, inserting the loading node and the unloading node in the best possible position. This is determined by the minimum additional costs. We do not consider the spot charter at this stage for the following reason. If there are orders that require a higher minimal additional cost to serve the order as compared to the spot charter's rate, all these orders will be put on the spot charter. However, when serving two or more of those 'far' customers together, it might result in a lower total cost.

This does not mean that the initial solution cannot contain orders on the spot charter. When we encounter contracted orders that cannot fit feasibly anymore in our available fleet in any possible position, then we assign the order to a spot charter. However, if the respective order concerns an optional cargo, we simply neglect it, thus we put it on a dummy ship. It is still possible to insert the neglected cargoes later

[1] Computational Infrastructure for Operations Research, http://www.coin-or.org

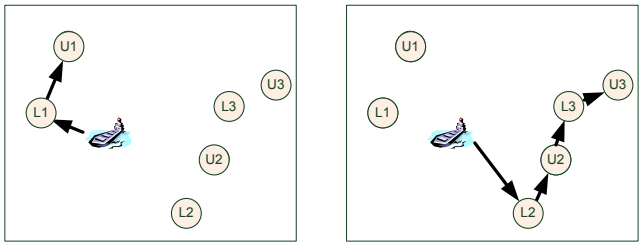


*Fig. 5.1:* Relevance of insertion sequence. The left picture shows the result if order 1 is inserted first. The right picture shows the resulting initial solution when either order 2 or 3 is inserted first.

during our search. Since we assume that an external spot charter is always available, this implies that a feasible initial solution always exists.

Since contracted orders are required to be served, they will be inserted first, after which the optional cargoes follow. The sequence of insertion is very important for the resulting initial solution. In [28], they applied this simple heuristic to insert orders one-by-one to generate a feasible solution to the PDPTW, but never mentioned how they determined the sequence of insertion. To illustrate the importance of the insertion sequence, consider the simple example in Figure 5.1. Here, we show two different feasible initial solutions being generated. Assume that due to time window constraints, the proximity and the fact that we have only one ship, we cannot serve all customers. The picture on the left depicts the result when first the order 1 is inserted. The remaining two orders cannot be inserted anymore in any position. Alternatively, when first order 2 or order 3 is inserted, the resulting initial solution is depicted in the picture on the right. In conclusion, the sequence of insertion is the main determinant in generating an initial feasible solution, when we apply our simple greedy heuristic.

In our case, we randomize the order of insertion, taking into account, however, that contracted orders will be scheduled prior to the optional ones. To account for the perils of random initialization, we run the TTS multiple times. This makes our TTS similar to a multi-start local search heuristic (e.g. [6]), which in turn *diversifies* our search, as each starting point would represent a different region in the solution space. This approach is efficient for problems where solutions can be easily constructed, but where it is difficult for the search to move between different search space regions. Ship scheduling problems have these characteristics, as they are often tightly constrained, and search algorithms can get trapped easily in local optima.

In addition to random initialization, we also deploy a deterministic circled sweep heuristic. This heuristic determines the sequence of insertion deterministically, by starting from the depot, and select the pickup order that is located closest to the depot first, followed by the second closest, until all pickup orders are inserted. All respective unloading nodes are inserted at the same time, paired up with the loading nodes. Note that we still insert all contracted cargoes first.

## *5.2 Neighborhood structure*

We defined two simple neighborhood moves for our TTS, a single paired insertion move and a paired swap move. During our search, we do not allow for infeasible solutions;

we do not allow moves that violate capacity or time constraints. Nor do we allow to neglect contracted orders. For tightly constraint situations, this might seem to put thick barriers around the neighborhood of the simple single paired insertion move. To this end, we allow for a paired swap move between two routes, which removes the respective orders from both routes prior to reinserting them. In most of the benchmark cases, these moves suffice to obtain quick and good solutions (see 6.1).

Additionally, we make use of *elite candidate lists* to speed up the search process. Generally, during every iteration, all possible moves are assessed and one is chosen to operate on the current solution. All other potential moves are discarded. We implemented an elite candidate list that records all moves that improve upon the current solution. The best move is on this list, which will be removed from the list if it is applied. Additionally, all moves that are not applicable anymore on the new solution, will be removed as well, i.e. all moves that involve removing orders from a ship, which route has been altered by the current best move. During the subsequent iterations, the remaining moves in the candidate list are considered and reevaluated, from which the best is chosen, until no improving moves remain. From the new obtained solution, the search resumes as normal, and another candidate list is generated.

### *5.2.1 Paired insertion move*

The simplest, yet most useful, move is the paired insertion move. It simply selects an order, be it an order that already is scheduled on a ship, an order scheduled for the tramp ship, or a neglected optional order on the dummy ship, and inserts it into another ship. It examines all possible selections and insertions of orders, considering both the respective loading and unloading nodes at the same time. Incremental changes in the objective value are calculated for each move, upon which the best move is selected.

When the best move is identified, the order is removed from its present route, and inserted in the best possible way in the new route. Note however, that it is also possible to insert the order in the same route as it were removed, allowing the paired insertion move to improve within routes as well.

Additionally, this move also allows for unscheduling optional cargoes. For example, if an optional cargo is scheduled on a route, but the revenue from serving the cargo is lower than the cost of the detour, it is more economical to simply neglect the optional cargo and remove it from the route. Since the move requires a new ship to put the order on, we have a dummy ship with a route of cargoes that represents all orders that are being neglected. Obviously, neglected optional orders on the dummy ship can also be taken and scheduled onto a real ship. Similarly, we have a dummy ship that represents all orders that are chartered out, which can be both contracted and optional cargoes.

The paired insertion move is demonstrated in Figure 5.2. The nodes with an L represent loading nodes, whereas the nodes with an U represent the respective unloading nodes. The dotted arrows indicate the legs that are being altered by this move. The red arrow represent the move that is conducted in this example, which is to move the order of ship 1 in position 3 to ship 2. Heuristically, the best possible insertion in ship 2 is made, which result in the highest profit, or equivalently, the lowest incremental cost of insertion. This is demonstrated in Figure 5.3, which demonstrates the move more intuitively in a Euclidean Space.

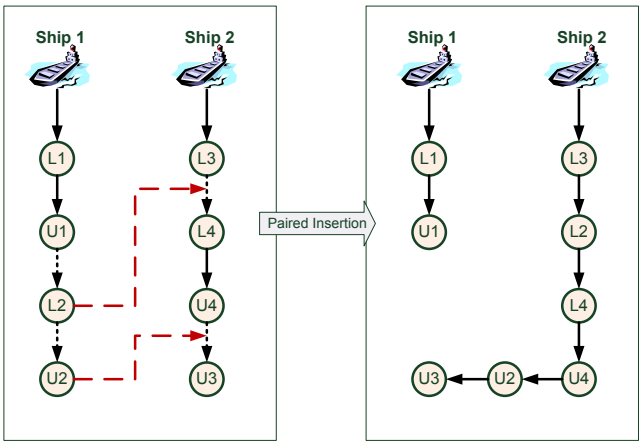


*Fig. 5.2:* The paired insertion move.

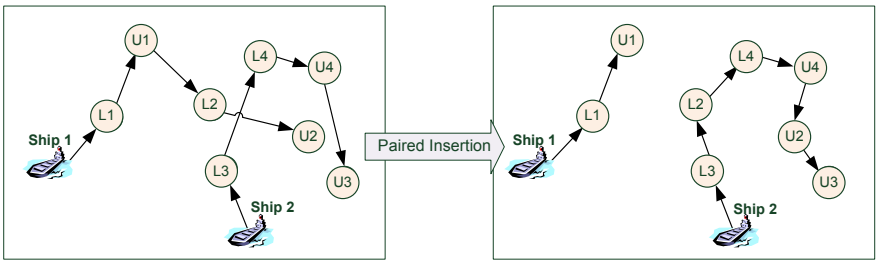


*Fig. 5.3:* The paired insertion move in a Euclidean Space.

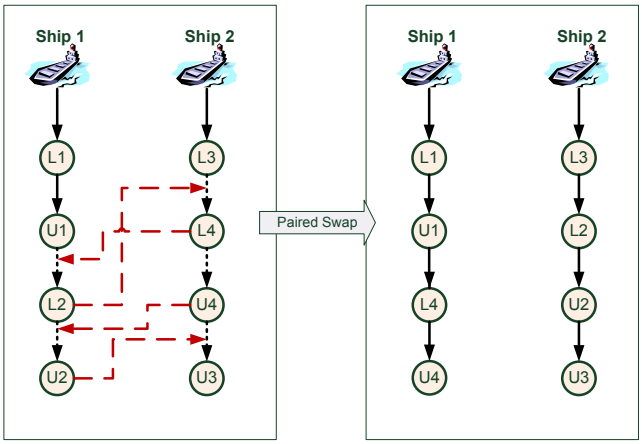


*Fig. 5.4:* The paired swap move.

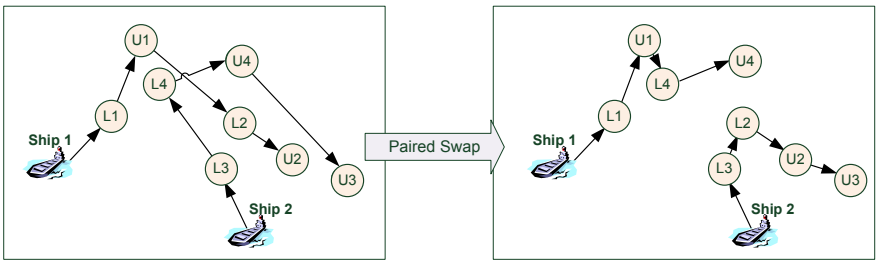


*Fig. 5.5:* The paired insertion move in a Euclidean Space.

### 5.2.2 Swap move

In tightly constrained situations, a single paired insertion move may get stuck. Consider for example a case where we have two ships that are both scheduled at full capacity, but the routes are not optimal (see Figure 5.5). The paired insertion move cannot move, since the moves are infeasible one at a time. The paired swap move, removes both orders first, swaps them, and reinserts the loading and unloading nodes in the best possible way. It is a more expensive move, since every possible swap needs to be considered, but in return, the search is moved into a new search space region. Swapping with the dummy ships is not allowed, nor is swapping within the same ship.

Figures 5.4 and 5.5 illustrate the paired swap move. We see from Figure 5.5 that before the move is conducted, the routes are clearly not optimal. Yet, using only the paired insertion move, the tabu search was unable to escape from this local optimum. Assume for this example that orders number 2 and 4 have really tight time window constraints, and serving both cargoes by a single ship is infeasible. Since we only allow feasible moves, the paired insertion move does not suffice in this case.

The tabu search algorithm makes primarily use of the paired insertion move. When an unimproving move is made, i.e. no more improving moves can be made using the paired insertion move, the algorithm switches to the paired swap move for one single

| Data set | | Swap move neighborhood | | | |
|---|---|---|---|---|---|
| | | **With** | | **Without** | |
| | | Best Profit | CPU time | Best Profit | CPU time |
| 100-cust | C1 | 828.94 | 4.12 | 828.94 | 2.05 |
| | C2 | 591.56 | 43.38 | 591.56 | 5.22 |
| | R1 | 1650.80 | 3.01 | 1687.88 | 1.72 |
| | R2 | 1482.59 | 22.64 | 1482.59 | 5.58 |
| Average | | 1138.47 | 18.29 | 1147.74 | 3.64 |
| Gap ratio | | 0.992 | 5.022 | 1 | 1 |
| 200-cust | C1 | 2704.57 | 26.37 | 2704.57 | 15.36 |
| | C2 | 2015.13 | 240.11 | 2015.13 | 30.79 |
| | R1 | 5081.59 | 45.09 | 4983.93 | 31.46 |
| | R2 | 5094.40 | 126.69 | 5217.00 | 40.23 |
| Average | | 3723.92 | 109.56 | 3730.16 | 29.46 |
| Gap ratio | | 0.998 | 3.719 | 1 | 1 |
| 400-cust | C1 | 7152.06 | 366.56 | 7337.07 | 187.45 |
| | C2 | 4505.90 | 4801.87 | 4903.55 | 432.69 |
| | R1 | 11850.59 | 417.36 | 12109.62 | 366.06 |
| | R2 | 11341.47 | 2339.44 | 11516.56 | 523.31 |
| Average | | 8712.50 | 1981.31 | 8966.70 | 377.38 |
| Gap ratio | | 0.972 | 5.250 | 1 | 1 |
| **Total Average** | | 11341.47 | 2339.44 | 11516.56 | 523.31 |
| **Gap ratio** | | 0.981 | 5.138 | 1 | 1 |

*Tab. 5.1:* Experiments with or without the swap move on different benchmark data sets. Using the swap move slows down the algorithm a lot, but in return reaches better optimal solutions.

iteration. This enables the search to move into a different search space and resume the search with the computationally less expensive paired insertion move.

### *5.2.3 Neighborhood experiments*

In this section we perform a number of experiments to assess the usefulness of the swap move, and the usefulness of an elite candidate list. It appears that applying the elite candidate list is very useful, as we gain a lot of speed without significant loss in solution quality. The swap move, on the other hand, reaches better solutions quality, but does so against an enormous amount of CPU time, relative to using only the paired insertion move.

To make sure randomness is no factor during the experiments, we ran the algorithm using the deterministic circled sweep initialization. The initial tenure is set to 15. The maximum iterations is set to 1000, but the algorithm will terminate the search when no new best solutions are found in 30 consecutive iterations. The different types of data sets are further explained in 6.1. The experiments are run on an Intel Core2 Duo, 2.1 Ghz and 2 GB Ram. For the swap move experiments we made use of the candidate list, and for the candidate list experiments we disabled the swap moves.

In Table 5.1 we report the experiments we ran to assess the usefulness of the swap move. For the smallest data sets of 100 customers, the swap move improves the solution quality only for the R1 instance. For other instances it does not gain

anything, but the algorithm becomes 5 times slower. Similar story for the 200-customer instances; no gain in return for tremendous CPU time, relatively to only using the paired insertion move. When we look at the bigger 400 customer instances, the swap move neighborhood manages to obtain better solutions as compared to not using it. It enables the tabu search to explore different areas. On average, solution quality can improve by 2% at a cost of almost 5 times the CPU time. In conclusion, we argue that our implementation of the swap move is ineffective.

The experimental results for using the candidate list are reported in Table 5.2. As we see in this table, the algorithm becomes much faster, especially for big cases. On the smaller data sets of 100 and 200 customers, making use of the candidate list for these experiments appeared dominating; at least as good solutions are found in less time. On the 400 customers data sets, the improvement in CPU time becomes even more apparent. This makes sense, since in bigger data sets, more useful moves can be performed at the same time, especially in the beginning of the run.

When the elite candidate list is applied, the moves selected each iteration may not be the best one available. Therefore, the use of such candidate lists would require generally more iterations to improve upon a solution relatively to the normal tabu search, when during each iteration the best move is selected. Despite the increasing number of iterations it requires, time wise it is still be much faster. On average, using the elite candidate list, we gain approx 50% in speed without a trade off in solution quality.

## 5.3 Tabu List

Every move that is performed is recorded on a central tabu list for a number of iterations, depending on the tenure value. Two move attributes are recorded, i.e. the id of the order that is being moved, and the ship that it is inserted in. If this move was not so good, and in the next move the nodes are removed again, this prevents the same node to return to the ship that it is inserted (which apparently was not a good idea anyway). It is allowed to return the nodes to the original ship, and in the following iterations, other (sub-optimal) moves will be considered.

We can also think of recording the current ship, as opposed to the new ship. The disadvantage of this approach however, appears when we consider the aforementioned example. When a move returns the node that was being moved, back into the original ship, it cannot be moved anymore, since the move is on the tabu list. However, since it was interesting to move this node away in the first place, we might want to consider inserting it on other ships.

Whenever a new best solution is found through a move that is on the tabu list, we override the tabu list by means of the aspiration criteria. Without such an aspiration criteria, we risk that better solutions are not found, for the required moves are banned on the tabu list. The point of the algorithm is to find the best solution possible, so therefore we should not prevent the search from finding a better best solution.

The tenure value, which is the parameter for the length of the tabu list, can be fixed to a certain value, or it can alter during the search, adapting a reactive tabu search. We implemented a reactive tabu search, which increases or decreases the tenure value based on the previous moves. If the previous two moves were improving moves, the tenure value is decreased by multiplying a factor $x$. If the previous move was an unimproving one, the tenure value is increased by a constant $y$.

In Tables 5.3, we present some experiments that examine the different possibilities.

| Data set | | Elite Candidatelist | | | |
|---|---|---|---|---|---|
| | | **With** | | **Without** | |
| | | Best Profit | CPU time | Best Profit | CPU time |
| 100-cust | C1 | 828.94 | 1.96 | 828.94 | 2.35 |
| | C2 | 591.56 | 5.39 | 591.56 | 7.25 |
| | R1 | 1687.88 | 1.72 | 1699.41 | 2.43 |
| | R2 | 1482.59 | 5.60 | 1535.39 | 6.88 |
| Average | | 1147.74 | 3.67 | 1163.82 | 4.73 |
| Gap ratio | | 0.986 | 0.775 | 1 | 1 |
| 200-cust | C1 | 2704.57 | 15.23 | 2704.57 | 18.40 |
| | C2 | 2015.13 | 30.51 | 2015.13 | 41.33 |
| | R1 | 4983.93 | 29.03 | 5441.70 | 27.46 |
| | R2 | 5217.00 | 39.88 | 5346.90 | 66.69 |
| Average | | 3730.16 | 28.66 | 3877.08 | 38.47 |
| Gap ratio | | 0.962 | 0.745 | 1 | 1 |
| 400-cust | C1 | 7337.07 | 179.99 | 7152.06 | 315.39 |
| | C2 | 4903.55 | 330.07 | 4808.30 | 727.32 |
| | R1 | 12109.62 | 269.42 | 12082.03 | 596.60 |
| | R2 | 11516.56 | 410.53 | 11723.26 | 872.41 |
| Average | | 8966.70 | 297.50 | 8941.41 | 627.93 |
| Gap ratio | | 1.003 | 0.474 | 1 | 1 |
| **Total Average** | | 4614.87 | 109.94 | 4660.77 | 223.71 |
| **Gap ratio** | | 0.990 | 0.491 | 1 | 1 |

*Tab. 5.2:* Experiments with or without a candidate list on different benchmark data sets. Using an elite candidate list speeds up the process significantly, especially for bigger and more complex cases, without significant loss of solution quality.

We performed these experiments on the more difficult data sets, i.e. those that have customers randomly located (denoted with an R), so that the differences are more apparent, since these 'puzzles' require more 'searching'.

For the experiments, we applied the deterministic circled sweep heuristic for the initial solution. We disabled the swap move, and we made use of elite candidate lists, and the maximum number of iterations is set to 1000. However, we put a stopping condition in place, which stops the search when for a consecutive number of iterations, no new best solution has been found. This number is double the tenure value, since the search starts cycling if it cannot seem to escape from a local optima using the present length of the tabu list. When cycling appears, there is no use in further searching. However, we inhibit a minimum of 10 iterations, in case the tenure value drops below 5 and stops prematurely.

In the upper table, we first performed some experiments using a fixed tenure value, ranging from 3 to 30. The best solution quality is obtained with the highest tenure, which consequently requires the longest computation time on average. When we look at the average gap ratios relative to the solution and CPU time obtained by using a tenure of 30, we see that speed and solution quality clearly is a trade off. In Figure 5.6, we made a scatter plot with on the x-axis the solution gap ratio and on the y-axis the CPU time gap ratio. The diamonds represent different parameter settings for the tabu list. The blue points represent the results obtained using a fixed tenure value. For example, by using a fixed tenure of 10, on average, we gain 45% in CPU time, but we lose over 3% in solution quality. Now if we state that we are willing to trade off 1% of the solution quality for a 20% gain in CPU time, everything down the blue line is an attractive option. Points on the right of the blue line is less attractive, since less gain is achieved with the trade off. Points on the left of the blue line are better 'deals'. We see in this scatter plot, that a fixed tenure of 15 has a nice balance between speed and solution quality. If we opt for better solution quality, and say we have ample time to run the algorithm, we can choose for a higher fixed tenure of 30. The other options of using a lower fixed tenure are not so good, if we take into account the results of using a reactive tenure.

The experiments with a reactive tabu list, presented in the lower table, are performed with an initial tenure value of 15, which appeared to be the best balance in the previous experiments. We see a similar trade off in solution quality and CPU time. The gap ratio values are calculated relative to the values obtained by using a fixed tenure of 30, as in the upper table, to maintain comparability.

The red points in Figure 5.6 represent the reactive tabu list with different parameter settings. The values are calculated relative to the big diamond on the (1,1), which is obtained using a fixed tenure value of 30. We do not see much significant improvements if we compare it to using a fixed tenure, as we had expected. For example, the red point very close the big diamond, which represents the parameters $x = 0.9$ and $y = 5$, is slightly faster with a slight trade off in solution quality. Nothing too significant. If we once again state that we are willing to trade off 1% in solution quality for a gain of 20% in CPU time, we draw the red line from the origin of the point that represents a fixed tenure of 15. Here we see an interesting diamond at (1.032, 0.293), representing the parameter setting of $x = 0.9$ and $y = 1$. It is the best option if we would like to come up with a solution quickly and we are willing to trade off some solution quality.

In conclusion, there is no best option; we have three best options. If the solution quality matters most, we opt for a fixed tenure of 30. When we need a good solution in very little time, we opt for a reactive tenure with the parameters $x = 0.9$ and $y = 1$.

**Fixed**

| Data set | | 3 | | 7 | | 10 | | 15 | | 30 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | BP | CPU | BP | CPU | BP | CPU | BP | CPU | BP | CPU |
| 100-cust | R1 | 1687.88 | 1.03 | 1687.88 | 1.16 | 1687.88 | 1.34 | 1687.88 | 1.70 | 1687.88 | 2.80 |
| | R2 | 1539.28 | 3.32 | 1511.41 | 4.22 | 1482.59 | 5.28 | 1482.59 | 5.55 | 1482.59 | 8.59 |
| 200-cust | R1 | 5341.67 | 8.34 | 5212.29 | 21.37 | 5336.62 | 15.86 | 4983.93 | 30.16 | 5108.26 | 32.78 |
| | R2 | 5240.23 | 28.93 | 5217.00 | 37.38 | 5217.00 | 35.36 | 5217.00 | 39.61 | 5007.46 | 74.82 |
| 400-cust | R1 | 12209.02 | 84.61 | 12122.78 | 197.19 | 12122.78 | 189.02 | 12109.62 | 272.19 | 11681.65 | 364.26 |
| | R2 | 11759.95 | 175.27 | 11694.68 | 240.34 | 11679.50 | 310.80 | 11516.56 | 410.28 | 11430.75 | 742.53 |
| **Average** | | 6296.34 | 50.25 | 6241.01 | 83.61 | 6254.39 | 92.94 | 6166.26 | 126.58 | 6066.43 | 204.30 |
| **Gap Ratio** | | 1.038 | 0.246 | 1.029 | 0.409 | 1.031 | 0.455 | 1.016 | 0.620 | 1 | 1 |

**Reactive tabu list**

| Data set | | x = 0.7 | | | | | | x = 0.9 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | y = 1 | | y = 3 | | y = 5 | | y = 1 | | y = 3 | | y = 5 | |
| | | BP | CPU | BP | CPU | BP | CPU | BP | CPU | BP | CPU | BP | CPU |
| 100-cust | R1 | 1687.88 | 1.26 | 1687.88 | 1.01 | 1687.88 | 1.10 | 1687.88 | 1.02 | 1687.88 | 2.21 | 1687.88 | 2.79 |
| | R2 | 1539.28 | 3.76 | 1486.91 | 5.24 | 1482.59 | 4.92 | 1539.28 | 3.20 | 1482.59 | 6.68 | 1482.59 | 7.40 |
| 200-cust | R1 | 5341.67 | 7.61 | 5336.62 | 17.70 | 5108.26 | 21.54 | 5212.29 | 16.55 | 5108.26 | 30.19 | 5108.26 | 35.38 |
| | R2 | 5240.23 | 26.12 | 5217.00 | 34.22 | 5217.00 | 31.74 | 5240.23 | 26.20 | 5217.00 | 42.09 | 5007.46 | 80.45 |
| 400-cust | R1 | 12209.02 | 73.82 | 12122.78 | 171.13 | 12122.78 | 190.96 | 12209.02 | 75.25 | 11689.58 | 558.40 | 11689.58 | 481.08 |
| | R2 | 11759.95 | 175.52 | 11694.68 | 252.25 | 11517.23 | 423.61 | 11697.75 | 237.37 | 11517.23 | 446.47 | 11504.35 | 552.39 |
| **Average** | | 6296.34 | 48.01 | 6257.64 | 80.26 | 6189.29 | 112.31 | 6264.41 | 59.93 | 6117.09 | 181.01 | 6080.02 | 193.25 |
| **Gap ratio** | | 1.038 | 0.235 | 1.032 | 0.393 | 1.020 | 0.550 | 1.033 | 0.293 | 1.008 | 0.886 | 1.002 | 0.946 |

*Tab. 5.3:* Experiments with different tabu tenure settings. The upper table reports the results of using different fixed tenure value, whereas the lower table reports the use of a reactive tabu list, with different parameter settings. Note that the values of the gap ratio are relative to the results obtained with a fixed tenure of 30.

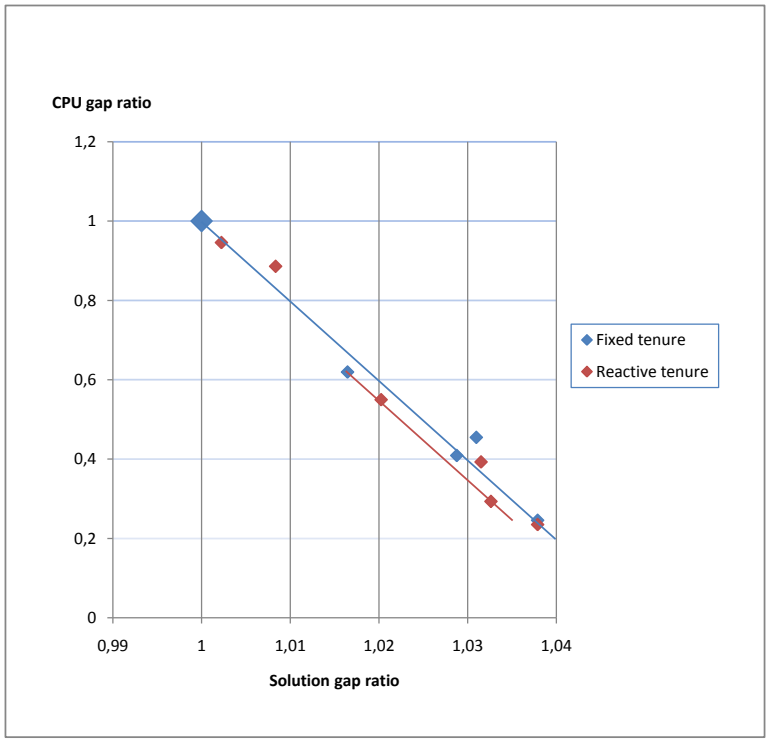


*Fig. 5.6:* Scatter plot of different tabu list parameter settings, along two performance measures, i.e. CPU time and solution quality. The values are obtained as a percentage relative to the values obtained using a fixed tenure of 30, depicted by the big diamond on point (1, 1).

For a good balance in between, use a fixed tenure of 15.

## 5.4 Overview of the TTS algorithm

The TTS algorithm runs for a predefined number of runs, which also determine the number of different initial solutions we generate to start the search from. We always generate one initial solution using the deterministic circled sweep heuristic. When we set the algorithm to run 5 times, this implies that one initial solution is generated using the circled sweep heuristic, and four others randomly. For every run, after a feasible initial solution is generated, the TTS algorithm starts running until a stopping condition is met, i.e. the maximum number of iterations is reached, or when for a subsequent number of iterations no new best solutions are found. During each iteration, all possible moves are generated and assessed, from which the best possible move is selected, if it is not on the tabu list. A new current solution is obtained for the next iteration. If we have a candidate list, we will pick the best remaining move from the candidate list, provided that it will improve the current solution. After the TTS is finished running, the best solution from all runs is chosen. The TTS algorithm is summarized in a flow chart in Figure 5.7.

## 5.5 Embedded Speed Optimization Heuristic

This section describes the speed optimization heuristic that we devised and embedded in the TSS model, resulting in the TSS-SD model. We opt to optimize not only the schedule, but also the sailing speeds in between the nodes on the schedule. As is discussed in Section 2.2, slow steaming offers a significant opportunity to save in operating costs. On the other hand, we like to keep our model pragmatic and computationally tractable. We first state the complications of embedding a speed optimization heuristic; Then, in Section 5.5.3, we give the description of our heuristic and how we embedded the heuristic in the tabu search.

### 5.5.1 Dual optimization

There are a few ways one can consider in solving this problem. One way is to optimize the sailing schedules first, and afterwards optimize the sailing speeds on this schedule. This simple approach has a major drawback that the found solution is most likely to be far from optimal. Since slow steaming decisions have a direct consequence for the feasibility region of scheduling decisions, and vice versa, we need to optimize both decision variables simultaneously. Refer to the test case depicted in Section 6.2.2 for a clear illustration of the importance of simultaneous optimization.

In the paper of [13], which studies the ship scheduling problem with soft time-windows, time is made discrete. On a fixed route, the arrival times at the specific nodes are then optimized, which in effect decides the cruising speed. For their approach, they made duplicate dummy nodes with each node representing a different arrival time. For example, when there are three possible times of arrival, which would correspond with slow steaming, medium steaming, and maximum speed, there will be three different nodes each representing a different decision. Once the network of nodes is generated, the shortest path is found using a dynamic programming algorithm. They used a set partitioning approach, with each column representing a feasible route, which is optimized using the aforementioned approach.

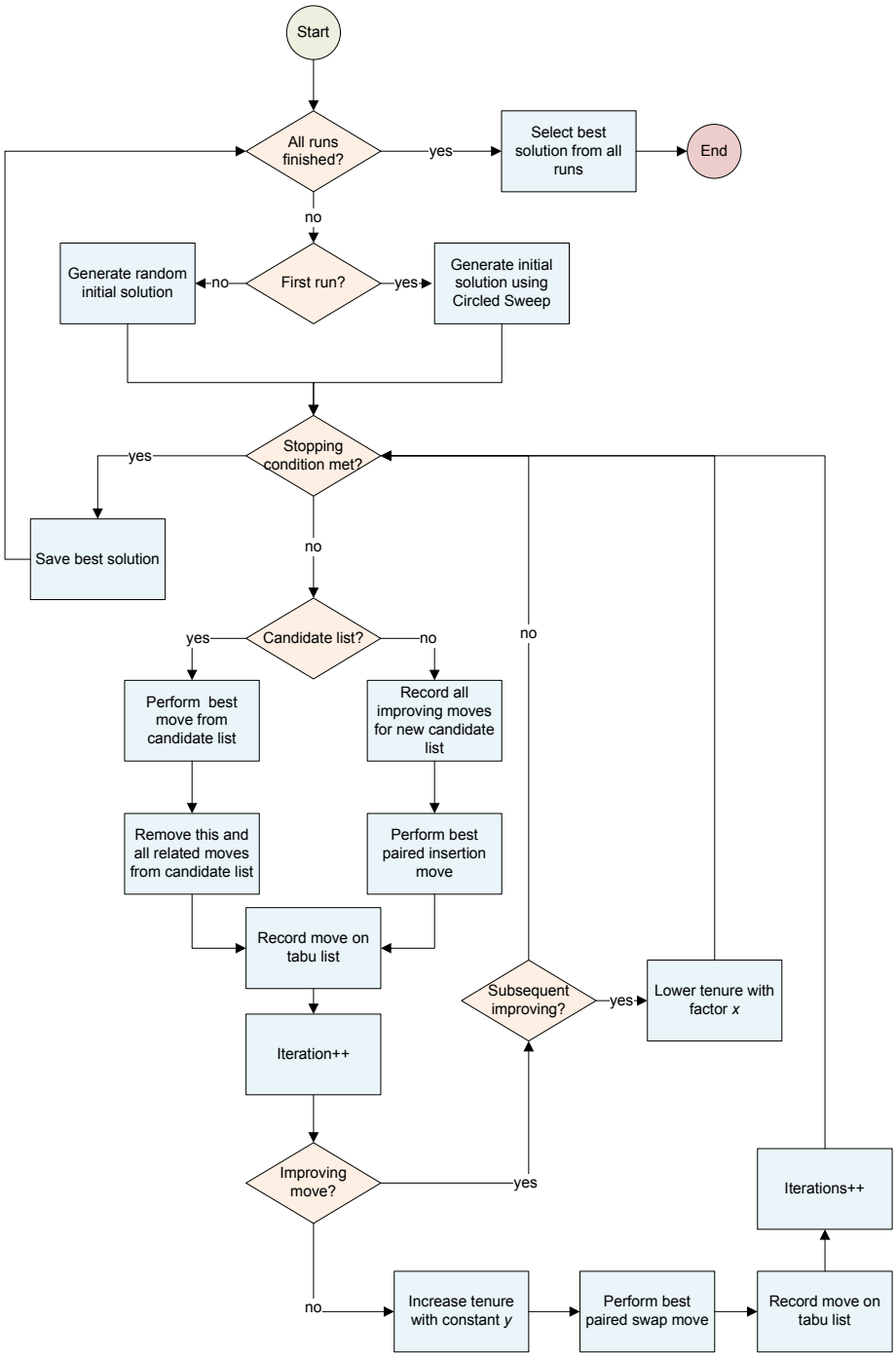


*Fig. 5.7:* Flow chart overview of our tabu search algorithm.

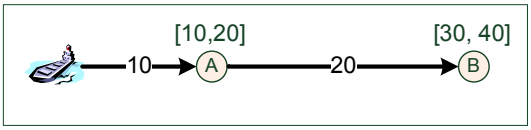


*Fig. 5.8:* A simple example of the optimization problem of sailing speeds on the legs within a schedule.

We argue that this approach is highly inefficient. The search space will grow exponentially with the number of possible discrete arrival times. When we would like to optimize sailing speeds to using smaller steps, the search space would explode, resulting in intractability for mid-sized problem. This issue has been addressed by [13] as well.

Now we both want to optimize scheduling decisions and sailing speed decisions simultaneously, but we do not want to enlarge the search space for the tabu search. Below, in Section 5.5.3, we present a fast heuristic that is embedded in the tabu search, without exploding the search space.

### *5.5.2 Speed optimization issues*

In order optimize both the schedule and its associated sailing speeds in between the nodes, we need to embed an optimization heuristic in the tabu search. Generally, as in the TSS model without speed decisions, the sailing speed is set to the maximum, which enables most feasible moves in the search space. In the TSS-SD model, we do the same, but all feasible moves are then optimized using a heuristic before they are being evaluated. This way, the resulting solution will implicitly have made the (sub)optimal trade off between slow steaming and additional cargo revenues, resulting in the highest profits.

The speed optimization heuristic requires to be fast, since every feasible move requires the heuristic to re-optimize up to two ship schedules. We designed an iterative process which lowers the speed of the most expensive leg by one unit $\delta_s$, and continue until no more slow steaming opportunities are feasible.

Each ship within a fleet may be optimized one by one, since there are no inter-relations between sailing speed of ships within a fleet. Within a single ship schedule, provided that it has at least one customer order, are multiple sailing legs. Each leg has an associated cost, which depends on the distance, the port costs (which we neglect), the fuel consumption of the ship, the fuel price, and the sailing speed. All parameters are fixed, except for the latter. Alternating the sailing speed on a specific leg will have direct consequences for both the operating costs and the arrival time at the next node. When we arrive later at the next node, this may not only imply that we might not make it to an additional customer order anymore, but it may also imply lost slow steaming opportunities on later legs. One has to think twice before applying a just-in-time approach.

Consider the following example depicted in Figure 5.8. We have a single ship that is scheduled to go to node A to pickup cargo, which takes 10 days at maximum speed from the present location, and then to deliver the cargo at node B, which takes another 20 days to reach from node A at maximum speed. Picking up cargo at node A is allowed between days 10 and 20. Delivery of the cargo at node B is expected between days 30 and 40. We assume that the maximum speed of the ship is 1 unit

per day, and that the associated cost per unit at maximum speed is $10. This means that when the ship sails at full speed, the total operating cost is $300.

This example has two legs, and looking at the time windows, it offers slow steaming opportunities. For the first leg, we could sail at half speed, which would lower the associated cost of $100 for the leg to only $12.50 according to Equation (3.16). Arrival time at node A would be day 20, which is just in time. However, we need to hurry and sail at max speed to make it just in time at node B. The operating costs for this leg amounts $200, which totals $212.50 for the transport. Initially, we had slow steaming opportunities on the second leg as well, but this has been off set by the fact that we arrived so late at node A, because we sailed at minimum speed on the first leg.

Now let us look at the situation where we exploit slow steaming on the second leg, rather than the first leg. We would sail at maximum speed to pick our cargo up, and arrive at node A on day 10, with an associated cost of $100 for the first leg. Since this is our final delivery of our schedule, we like to arrive just in time at node B. We can arrive 30 days later. Sailing at two-third of maximum speed would optimally exploit slow steaming in this case. The resulting cost of the second leg is $59.25, and the total operating cost amounts $159.25. This is much lower than if we would exploit the first leg.

The reason for this is that by lowering the speed on the most expensive leg, it will result in the highest savings. Since the second leg is more expensive, due to the distance, it was a better option to slow steam on this leg. However, sailing slower has a direct effect on the cost of the leg, hence at some point the second leg turns cheaper than the first, and it would be more profitable to lower the speed on the first leg.

The optimal lies in somewhere between. With a precision of whole percentages of maximum speed, the optimal speeds are 88% and 70% respectively, with a total cost of $136.75. The heuristic that we used to find this optimum efficiently is described below.

### 5.5.3 Speed optimization heuristic

The subproblem of optimizing sailing speeds of a single route is an LP-problem on itself. To address an LP-solver to solve this subproblem would be computationally intractable, considering the amount of subproblems we need to solve. Every possible move would require calling the solver. To speed up the process, we devised an iterative heuristic, which lowers the speed on the most expensive leg of a specific ship schedule with $\delta_s$, and updates the cost of the leg. We select the most expensive leg, because slow steaming on this leg will result in the most significant savings. We continue this process until no more feasible slow steaming opportunities remain.

When we select $q$, which is the leg with the highest fuel consumption, we check whether this leg is feasible to lower the speed with $\delta_s$. This check is performed by calculating all the consequences of arriving later at the node it is sailing to, including the subsequent nodes it is going to visit with their currently set speeds on the respective legs. When lowering the speed on leg $q$ with $\delta_s$ results in arriving too late at any of the following node, the leg is infeasible to lower. The next most expensive leg that is feasible to lower is selected instead. When no more slow steaming opportunities remain, no leg is selected and the heuristic stops. Note also that the sailing speed cannot go lower than half of the maximum speed.

The pseudo code for optimizing a single route with $N$ legs is given in the box below. Note that CPNM stands for the cost per nautical mile at maximum speed. This value is dependant on the fuel price, the maximum speed and the daily fuel consumption of

the respective ship.

```
for i = 1 to N do
  leg[i] ← maxSpeed
  fuelCons[i] ← distance[i] * CPNM
end for
feasibleToLower ← true
while feasibleToLower do
  q ← Select most expensive leg that is feasible to lower with δ_s
  if q = null then
    feasibleToLower ← false
  else
    speed ← leg[q].speed
    leg[q].speed ← speed - δ_s
    fuelCons[q] ← distance[q] * CPNM * speedFactor(speed) {Eq. 3.16}
  end if
end while
```

## 5.6 Summary

This chapter presented the tabu search algorithm that we applied to solve our TSS model. The choices we made regarding some parameters are underpinned with small experimental data. There is no single set of optimal parameters; the experiments provides us with insight regarding some parameter options. Generally, the trade off is between computational time and solution quality. We studied in this chapter which parameters should be altered in order to get the best trade off.

We showed that the use of swap moves appeared expensive, but not so efficient in our implementation. We also assessed the use of elite candidate lists, which decreased the computational effort enormously, without a major loss in solution quality. We furthermore compared using a fixed tenure versus a reactive tenure, and came to the conclusion that the performance between the two does not differ significantly. We identified three options one could choose from to get the best trade off between additional computation time and solution quality. Generally speaking, the best parameter to alter, i.e. the best trade off is to be found, is the tenure value. Changes in the tenure value, either using a fixed or reactive tabu list, result in a 1 percent decrease in solution quality for the gain of 20 percent CPU time. Note that this implies that if we would want to increase the solution quality with 1 percent, we would require an increase of 25 percent CPU time. We will see in Section 6.1, that we can also increase the solution quality by increasing the number of runs, i.e. the number different initial solutions generated.

In addition to the basic TSS model that we developed for the Tramp Ship Scheduling problem, we devised the TSS-SD model in Section 5.5, which also optimizes the sailing speeds between the legs of the schedule. It is important to optimize sailing speeds simultaneously with the ship schedules, or otherwise risk reaching a solution far from the optimum. Section 5.5 addressed some issues related to optimizing sailing speeds simultaneously with ship schedules. We also state the importance of computational tractability for our TSS-SD, regarding the dynamic characteristic of the

industry.

Tackling these problems, we have developed an embedded heuristic to optimize single routes prior to evaluating a single move during the tabu search. This embedded heuristic sets all speeds to the maximum, and lowers the speed of the leg with the highest fuel consumption with a constant, until no more feasible slow steaming opportunities exist. This iterative procedure ensures (near)maximum savings while solving the subproblem quickly. Lowering the size of $\delta_s$, which is the step size of lowering speed, will increase the solution quality, but would require more computation time as well.

Now that we have developed the two models, TSS and TSS-SD, we would like to compare them quantitatively with each other. The quantitative comparison between the two models is presented in Section 6.3. First, we verify both models in the first two sections of the following chapter.

# 6. QUANTITATIVE ANALYSIS OF SLOW STEAMING OPPORTUNITIES

This chapter describes the computational study of this research, which includes a descriptions of the test case data, and the different experimental setups we used. First, we present the results of the benchmark verification study of our tabu search algorithm for the Pickup and Delivery Problem with Time Windows (PDPTW) by running our algorithm on benchmark data of [26] in Section 6.1. This allows us to compare our algorithm to the literature to some extend, since benchmark data for the TSS problem are not available. Secondly, we present very simple test case data sets to verify our TSS and TSS-SD models in Section 6.2, followed by the experimental setup and results in Section 6.3, which compares the performance of the TSS model with the TSS-SD model quantitatively on numerous test cases.

## *6.1 Benchmark study*

First we would like to verify our tabu search algorithm on the well-studied PDPTW, and benchmark it to the state-of-the-art literature. To this end, we used the benchmark data of [26] for its practically sized test cases. The benchmark data consist of numerous test cases that differ in three parameters, i.e. number of orders, tight or loose time windows, clustered or randomly located. The test cases are split on the number of customers: 100, 200, 400, 600, 800 and 1000 customers. Each set consist of six types of problems, i.e. (C1,C2,R1,R2,RC1,RC2). The $C$ stands for clustered, whereas the $R$ data sets are randomly located and $RC$ is a combination of both. The numbers represent how tight the time windows are, with 1 being tight, and 2 being loose. Each type of problem has eight to twelve instances per class of problems. The results are compared with the results reported in [26][1].

The experiments make no use of the swap move. The elite candidate list is deployed, and the initial tenure is set to 15. The parameters for the reactive tenure are set to $x = 0.9$ and $y = 1$. Maximum number of iterations is set to 1000, but a stopping condition is used, which stops the algorithm when for a consecutive number of iterations no new optimal solutions are found. This number is double the tenure value with a minimum of 10.

Experimental results for the 100-customer instances are presented in Table 6.1. Note that we aggregated all instances in six groups, and reported the average values. For instance, the C1 data set is comprised of 9 instances of clustered orders with tight time windows. We ran multiple experiments with different number of runs, ranging from 5 to 200, i.e. number of different randomly generated initial solutions from which

[1] The data sets and the results are obtained through the web site: http://www.top.sintef.no/vrp/benchmarks.html

the search proceeds. Obviously, the CPU time is (almost) linearly related to the number of runs performed.

As we can see, reading from Table 6.1, by using only 5 runs we obtain a solution on average within 20 seconds, but the solution we get is on average more than 6% away from the best known optimal solution. When we set the number of runs to 10, the second diamond in the figure, we improve the solution quality with 1.5%, and decrease the gap ratio to 4.5%. This would take double the time as compared to doing only 5 runs. The slope of the graph in this figure is decreasing, however, so in order to gain another 1.5% increase in the solution quality, we need to set the number of runs to 25 instead of doubling it to 20. The more runs, the better the solution quality gets, but in a decreasing fashion. It is up to the user to decide for how long the algorithm needs to search.

We also observe that our algorithm has the most trouble with the R2 problems, which are randomly located problems with wide time windows. Using 5 runs, the solution is obtained in 40 seconds on average, but the gap is on average more than 12%; even after 200 runs, running on average more than 1500 seconds, this gap is still almost 3%. For the other four categories of problems, our algorithm performs efficiently. For the C problems, we obtain just as good, if not even better solutions with only 10 runs. Note that these solution qualities do not improve anymore with the number of runs; apparently, (a very good) optimality has been reached.

In Figure 6.1 we depict the trade off between the number of runs and the solution quality. Here, the average gap ratio relative to the best known solution in [26] is depicted on the y-axis, and the average time in seconds is depicted on the x-axis. Each diamond represents the results corresponding to a different setting of number of runs. The first diamond represents 5 runs, i.e. one run which takes the initial solution generated by the circled sweep heuristic as the starting point, and four others that are generated using a random sequence of insertion (refer to Section 5.1 for more detail on this).

This section demonstrated our algorithm on large PDPTW benchmark instances of [26], data sets consisting of 100 customers. On average, the solutions we obtained were not as good as the best known optimal solutions reported by [26], but the algorithm managed to obtain very close to optimal solutions in very little time. With the dynamic tramp industry in mind and the practicality of the algorithm, we welcome this property. We also showed that the user can opt to run the algorithm as long as he desires, by increasing the number of initial solutions, in order to improve the solution quality.

## *6.2 Verification test cases*

The previous section verified our model to solve PDPTW problems. However, the scope of our research are TSS problems; this section presents some small test cases to verify our models on TSS problems. The first test case verifies the model's efficient use of the spot charter, which is an element not present in the PDPTW problem. Secondly, we verify our TSS-SD model in which we optimize sailing speeds. The according test case demonstrates the trade off of slow steaming versus additional revenue. In the third test case, we combined the first two test cases into a combined test case, and used two ships instead of only one. The latter test case demonstrates all our TSS-SD model's features on a very small scale.

100 customers

| Data set | Li&Lim | | 5 Runs | | 10 Runs | | 25 Runs | |
|---|---|---|---|---|---|---|---|---|
| | Profit | CPU (s) | Profit | CPU (s) | Profit | CPU (s) | Profit | CPU (s) |
| C1 | 832.09 | 225.56 | 834.85 | 6.44 | 827.27 | 12.66 | 827.22 | 29.20 |
| C2 | 589.23 | 196.25 | 597.63 | 29.17 | 589.93 | 60.09 | 589.86 | 140.39 |
| R1 | 1222.20 | 371.08 | 1271.61 | 6.19 | 1252.86 | 11.60 | 1250.61 | 26.13 |
| R2 | 973.87 | 1876.36 | 1094.56 | 38.79 | 1074.82 | 79.59 | 1042.35 | 192.15 |
| CR1 | 1387.60 | 261.25 | 1502.50 | 5.55 | 1465.81 | 10.80 | 1443.79 | 26.42 |
| CR2 | 1187.82 | 1536.13 | 1276.39 | 27.98 | 1257.01 | 52.58 | 1218.92 | 133.14 |
| **Weighted Avg** | 1039.02 | 769.14 | 1104.02 | 18.94 | 1085.80 | 37.79 | 1070.33 | 90.89 |
| **Gap Ratio** | 1.00 | | 1.063 | | 1.045 | | 1.030 | |

100 customers

| Data set | 50 Runs | | 100 Runs | | 150 Runs | | 200 Runs | |
|---|---|---|---|---|---|---|---|---|
| | Profit | CPU (s) | Profit | CPU (s) | Profit | CPU (s) | Profit | CPU (s) |
| C1 | 827.09 | 56.38 | 826.98 | 119.25 | 826.98 | 196.26 | 826.98 | 234.43 |
| C2 | 589.45 | 279.64 | 589.45 | 574.01 | 589.45 | 842.24 | 589.45 | 1122.66 |
| R1 | 1241.65 | 51.05 | 1239.09 | 106.38 | 1238.72 | 152.17 | 1237.85 | 202.07 |
| R2 | 1029.87 | 369.84 | 1018.42 | 760.69 | 1007.40 | 1180.64 | 1001.04 | 1527.69 |
| CR1 | 1435.44 | 50.76 | 1408.81 | 100.69 | 1400.30 | 147.75 | 1397.90 | 196.21 |
| CR2 | 1204.90 | 247.97 | 1192.26 | 498.82 | 1187.40 | 733.81 | 1183.49 | 967.22 |
| **Weighted Avg** | 1062.69 | 175.27 | 1054.26 | 359.03 | 1050.11 | 542.32 | 1047.77 | 707.64 |
| **Gap Ratio** | 1.023 | | 1.015 | | 1.011 | | 1.008 | |

*Tab. 6.1:* Performance of our algorithm compared to the results of Li&Lim on the 100 customers benchmark data of [26]. The values reported are average values for each of the six data set groups. Note that a direct comparison between our CPU time and the CPU time of Li&Lim is unfeasible, because they used a i686 system in 2001.

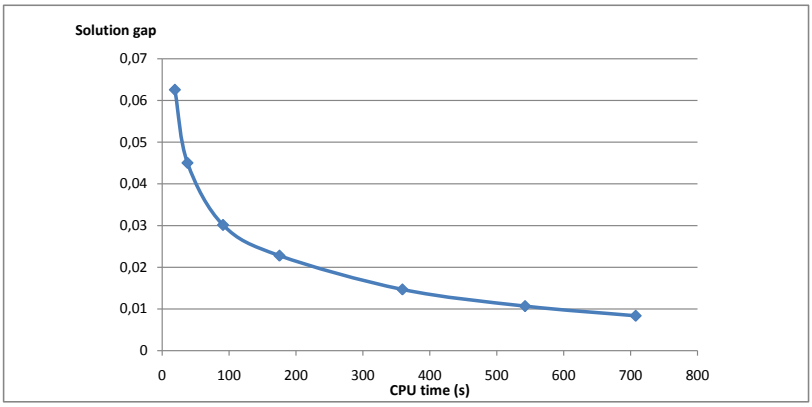


*Fig. 6.1:* Solution quality trade off with CPU time. The points represent from left to right 5, 10, 25, 50, 100, 150 and 200 runs respectively. The solution gap is calculated relative to the best known solution found by [26].

### *6.2.1 Test case: Spot charter*

This very simple test case demonstrates the models' use of spot charters in case of under capacity. A situation of under capacity is depicted in Figure 6.2, which also shows the associated optimal schedule. Here we have only one ship, and 3 orders. The orders are located in such a way, that not all three orders are able to be served within the given time windows by a single ship. If the ship decides to serve order 1, it will be too late for orders 2 and 3. Similarly, if the ship decides to serve orders 2 and 3, order 1 cannot be served anymore. The location of the ship is (10,10), and the constant sailing speed is 1 unit per day, at a cost of $10 per unit. The costs of hiring an external spot charter amounts $100, and the revenue of servicing a customer order is $200. The details of the orders are as follows:

| Order | X-coord | Y-coord | Early TW | Late TW |
|---|---|---|---|---|
| L1 | 10 | 13 | 3 | 5 |
| U1 | 9 | 13 | 5 | 10 |
| L2 | 10 | 7 | 3 | 5 |
| U2 | 12 | 7 | 5 | 10 |
| L3 | 14 | 7 | 8 | 12 |
| U3 | 15 | 7 | 8 | 12 |

When all orders are served by external spot charters, the total profit would be $300. If the ship decides to serve order 1, and let the other orders be carried by spot charters, the total profit would be $360 (since we sailed 4 distance units at a cost of $10 each), which is a better option. Alternatively, if the ship takes orders 2 and 3, and charter out order 1, the total profit is $420, hence the best option. This is indeed the optimal option that our model returns.

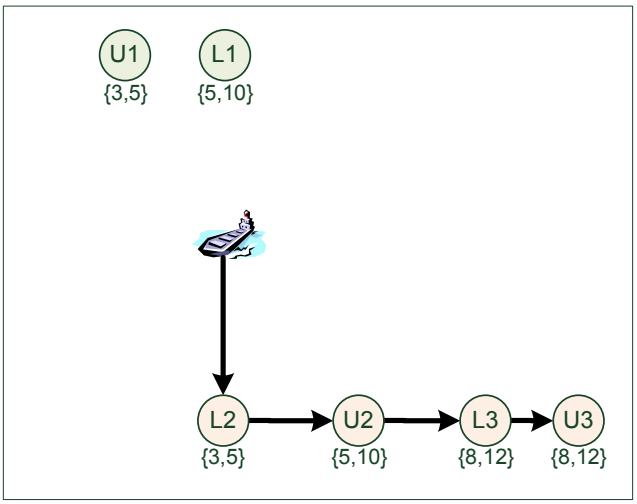


*Fig. 6.2:* Test case: Spot charter; a situation of under capacity. The optimal schedule is shown with the arrows. Order 1 is colored green to depict that it is being served by a spot charter. The numbers in curly brackets below the nodes depict the earliest and latest time of loading or unloading.

The previous example demonstrated the use of a spot charter for cases of under capacity. A spot charter is also useful if it is more economical to deploy it for certain very distant orders, rather than serving it with our own ships. For instance if in the previous test case order 1 was moved away near point (0,0), but the time window would be wide enough for our ship to make it after serving orders 2 and 3, it would still cost us more to sail all the way over there, than to hire a spot charter. We do not include another test case for this situation for its apparentness.

### *6.2.2 Test case: Speed Decisions*

The second test case is used to verify the TSS-SD model and shows the use of deciding on sailing speed. The location of the ship is (10,10), and the maximum sailing speed is 1 unit per day, at a cost of $10 per unit. The spot charter is priced at $200 per order. The revenue of servicing a customer order is $200. The details of the orders are as follows:

| Order | X-coord | Y-coord | Early TW | Late TW |
|---|---|---|---|---|
| L1 | 20 | 10 | 10 | 20 |
| U1 | 25 | 10 | 15 | 35 |
| L2 | 35 | 10 | 25 | 50 |
| U2 | 40 | 10 | 30 | 60 |
| L3 | 50 | 10 | 40 | 40 |
| U3 | 55 | 10 | 45 | 45 |

This test case has three non-contracted, optional orders. The first two orders have wide time windows, which enables slow steaming opportunities. However, order

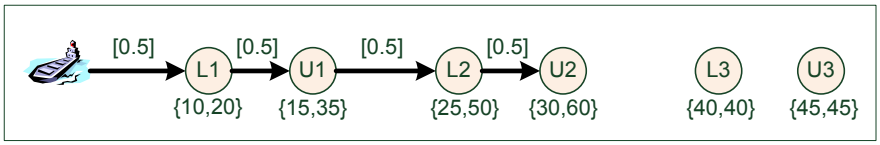


*Fig. 6.3:* Test case: Speed decision; a situation when neglecting an optional order results in a higher profit when making use of slow steaming. The optimal schedule is shown with the arrows; the optimal sailing speeds on the legs are shown on top of the arrow.

number 3 has no time window at all, and can only be reached if our ship sails at maximum speed.

Obviously if all cargo is served by the spot charter, the shipping company would make negative profits, as all revenues are offset by the higher costs of the spot charter. We would be better off neglecting all orders, since all orders are optional. The initial solution that is generated results in serving all customers (in order 1-2-3) at maximum speed, since we prefer as much cargo to be served by our own ships as possible for the initial solution (refer to Section 5.1). The total revenue amounts \$600. The total sailing cost is equal to \$450, hence the profit is \$150 when we serve all three cargoes, which would be the optimal solution if we used the TSS model, which sails at maximum speed at all times.

Now using the TSS-SD model, we can optimize sailing speeds between the nodes and save in operating costs. The consequence of doing this, however, is that we cannot reach order 3 in time and lose some revenue. When the savings in costs are higher than the lost revenue, this is profitable.

When we decide to serve only orders 1 and 2, we can sail at minimum speed, which is half the maximum speed (0.5 units/day), and make order 2 just in time. The corresponding sailing costs amount \$37.50 (using Equation (3.16)) compared to \$300 when we would sail the same distance at maximum speed. The total revenue is \$400, with costs as low as \$37.50, results in total profits of \$362.50. This is the optimal route depicted in Figure 6.3, which is also provided by our TSS-SD model.

This test case demonstrates that neglecting additional revenue, i.e. serving customer 3, can result in higher profits through more significant savings in operating costs, i.e. exploiting slow steam opportunities. The resulting profits of the TSS-SD model in this case is more than double the optimum obtained by the TSS model.

### *6.2.3 Test case: Fleet*

This slightly larger test case verifies the model using two ships and 6 orders. The optimal solution will require neglecting optional orders as well as the use of the spot charter. The location of the first ship is (10,10), and the second ship starts at (20,10); the maximum sailing speed is 1 unit per day, at a cost of \$10 per unit. The spot charter is priced at \$200 per order. The revenue of servicing a customer order is \$200. The details of the orders are as follows:

| Order | X-coord | Y-coord | Early TW | Late TW | contracted |
|---|---|---|---|---|---|
| L1 | 20 | 10 | 10 | 20 | false |
| U1 | 25 | 10 | 15 | 35 | false |
| L2 | 35 | 10 | 25 | 50 | false |
| U2 | 40 | 10 | 30 | 60 | false |
| L3 | 50 | 10 | 40 | 40 | false |
| U3 | 55 | 10 | 45 | 45 | false |
| L4 | 20 | 20 | 10 | 18 | false |
| U4 | 25 | 20 | 15 | 25 | false |
| L5 | 35 | 20 | 25 | 40 | false |
| U5 | 40 | 20 | 30 | 50 | false |
| L6 | 50 | 20 | 40 | 45 | true |
| U6 | 55 | 20 | 45 | 50 | true |

Note that this test case is similar to the second test case, but in duplicate and with two ships. The difference is that order number 6 cannot be neglected anymore, since this is a contracted order. Also, the time windows are slightly different, so that the simply slow steaming at minimum speed is not feasible anymore. The optimum needs to be found somewhere in between; this also demonstrates the importance of the step size $\delta_s$.

When we solve this test case using the TSS model, we obtain a profit of \$300. The corresponding schedule is that ship 1 services orders 1 to 3, and ship 2 services orders 4 to 6. Since we do not alter sailing speeds, this is set at the maximum of 1 unit per day.

When we solve the case using the TSS-SD model, we would neglect order 3 (similar to the previous test case) and serve order 6 using a spot charter, since the latter is a contracted order and may not be neglected. When we use $\delta_s = 0.1$, which denotes the step size of lowering the sailing speed, we obtain an optimal profit of \$687.45. Ship 1 would sail at minimum speed, but the optimal speeds on the legs of ship 2 would be 0.6, 0.8, 0.6 and 0.5 respectively. However, if we use $\delta_s = 0.01$, the optimal speeds of ship 2 would become 0.6, 0.75, 0.6 and 0.5, allowing us to slow down a little more on the second leg, resulting in the total profits of \$691.96. The drawback of using a smaller step size is that the embedded speed heuristic needs to iterate almost ten times as much. For large cases this might become an issue, but for this test case, this meant no significant difference in computation time. The optimal schedule is depicted in Figure 6.4, which is also what our TSS-SD model returns in three iterations.

## 6.3 Quantitative effects of Speed Decisions

This section will assess the quantitative effects of incorporating speed decisions into the TSS model. We compare the performance of the regular TSS model with the TSS-SD model on a few simulated cases described in Section 6.3.1. The two performance measures are solution quality and computation time. The computational results are reported and discussed in Section 6.3.2.

### 6.3.1 Experimental setup

This section describes how we generated test cases that we used to simulate a typical ship scheduling problem that a bulk shipping company faces. Below, we describe how we estimated the parameters of the model to match reality to some extend. We opt to

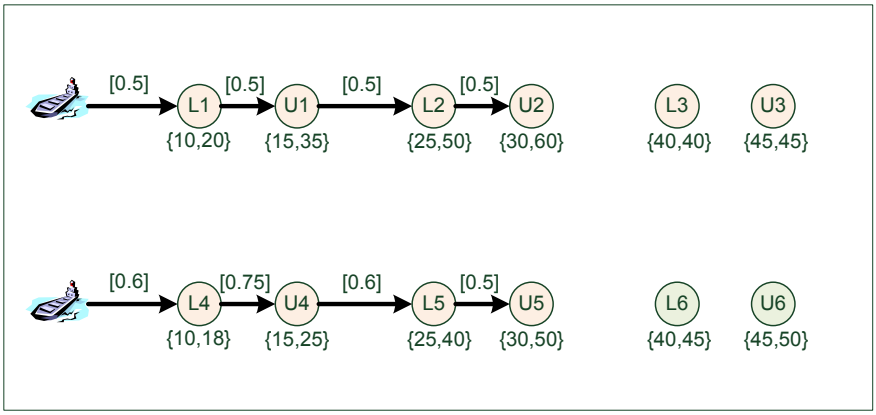


*Fig. 6.4:* Test case: A small test case with two ships and 6 orders. The optimal schedule requires the use of slow steaming as well as the spot charter and neglecting orders.

match realistic numbers, since the significance of slow steaming depends on the relative proportions of (sailing) costs and revenues. Furthermore, it would give us some, if not exact, practical insight in the range of the potential increase in profits in USD.

*Model parameters estimation*

The dry-bulk trip charter rates (of iron ore from Brazil to China) in 2007 ranged from $35.50 per tonne at the beginning of the year to $86.35 per tonne at the end of the year [35]. For our experiments we simplify this to range from $50 per tonne to $100 per tonne, which represents the range of the revenues rates created from transporting cargo.

The global average vessel size of bulk carriers in deadweight, i.e. maximum tonnes that a ship can carry, were at the end of 2006 about 80,000 tonnes, whereas at the end of 2007 this was about 86,000 tonnes. For our experiments we range the capacity of the ships between 25,000 and 150,000 tonnes. We randomize the weight of each order between 5,000 tonnes and 120,000 tonnes.

The time it requires to load or unload an order at a port is dependant on the speed of the crane and the quantity of the cargo. Dry bulk cranes' speed ranges from 1,000 tonnes per hour to 10,000 tonnes per hour[2], which is the range in which we randomize. Additional to the actual loading and unloading, a ship that is visiting a port requires some fixed time to enter the port, setup the crane and to exit the port. We estimate the total fixed time per port visit to be 3 hours.

We found a few bulk carriers in online catalogues[3] with reported deadweight and their sailing speed and fuel consumption. These numbers are shown in Table 6.2 and gives us a realistic feeling of these parameters. The fuel consumption increases with the size of the ship, but the sailing speed remains constant around 14 knots, which is about 26 km/h, with one knot equal to 1.852 km/h, i.e. one nautical mile per hour.

[2] http://www.nauticexpo.com (accessed November 2009)
[3] e.g. http://www.apolloduck.cn and http://www.alibaba.com (accessed October 2009)

| Ship Nr (D.W.T.) | Capacity (knots) | Speed (Tonnes/day) | Fuel consumption |
|---|---|---|---|
| 1 | 25,000 | 13 | 21 |
| 2 | 30,000 | 13 | 23 |
| 3 | 54,500 | 14 | 30 |
| 4 | 70,000 | 14 | 33 |
| 5 | 150,000 | 14 | 45 |

*Tab. 6.2:* Features of a few actual example bulk carriers, except for the last one with deadweight tonnage of 150,000, we made that one up as we could not find any information about ships of this size. This list also represents the list of optional ships for our fleet.

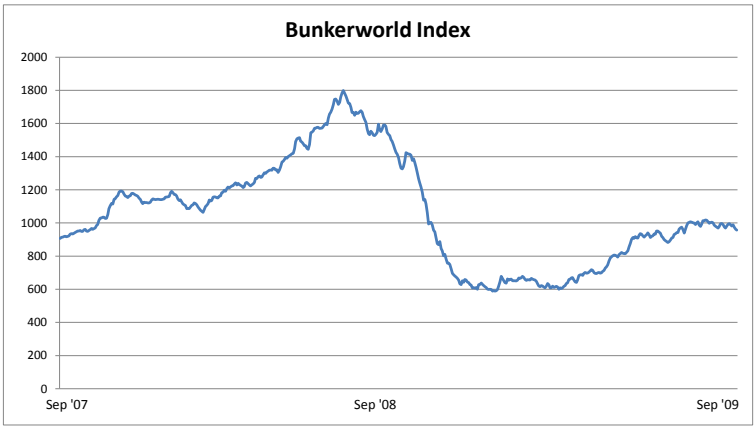


*Fig. 6.5:* Plot of the Bunkerworld Index of the last two years. This index depicts the global movement of bunker fuel prices in US$ per tonne.

The global bunker fuel prices are captured in the Bunkerworld Index (BWI), which is a weighted daily index made up of 20 key bunkering ports[4]. The index is denoted in USD per metric tonne, delivered on board basis, i.e. including barge transport and/or ex-pipe fees. As we see in Figure 6.5, which depicts the BWI from the past two years, bunker fuel prices are highly volatile. For our experiments, we take the rounded value of $1,000 per tonne, which is slightly less than the average price of the last two years, and is the most recent price level of bunker fuel.

A single voyage between Brazil and China takes on average about 37 days compared with a single voyage from Australia to China, which takes about 15 days; the trip from Beijing to Hamburg lasts for 30 days[35]. The sailing distance between Brazil and China is approximately 10,000 nautical miles. These numbers give us an idea of the distances that the bulk carriers sail. Our Euclidean space has a maximum size of 10,000 nautical miles, and distances between a pickup and a delivery node has a minimum of 3,000 nautical miles.

[4] http://www.bunkerworld.com/prices/index (Accessed October 2009)

The sizes of the test case studies performed in [23, 6, 7] range from 14 to 75 contracted orders and 4 to 20 optional orders. The fleet sizes range from 4 ships to 19, and the planning horizon ranged from 15 to 120 days. These test cases were derived from a real shipping companies, hence are practically sized. From all cargoes, on average, about 25% are spot cargoes. For our test cases, we used similar sizes and proportions. Regarding the specifications of the time windows, such as the general size and also the spread between the pickup and delivery, we could not find any values in literature. Therefore, we randomized the size of the time windows, while taking into account the feasibility. We do not allow the delivery time window to be earlier than its respective pickup delivery for example. Also, the latest possible day of delivery has to be at least reachable from its pickup location at maximum speed.

*Test cases*

Our test cases range in number of orders, both contracted and optional, number of ships and the planning horizon. While maintaining the proportions used in [23, 6, 7], we created three different classes of test cases, i.e. small, medium and large. The small test cases have 4 ships and 20 orders. The medium sized test cases use a fleet of 8 ships and have 40 orders. The large test cases use a fleet of 15 ships with 100 orders.

Within each class of test cases, we distinguish also in the planning horizon. When this horizon is small, this means that there are more orders in a smaller period of time. Conversely, a wider planning horizon implies a spread out of the orders over time, which means that more orders can be served one after the other. We consider a tight, normal, and wide planning horizon, i.e. 40, 100 and 200 days.

In total we have 9 different types of test cases, each of which we generate 8 instances of, totalling 72 randomly generated test case using the parameters described in the previous section. We depict each data set using a prefix in their name, e.g. mw3 denotes the third out of the eight test cases that are generated with 40 orders and a planning horizon of 200 days.

Every test case that we generated as described above consists only of the orders. Obviously, we also need a fleet which serves the orders. Unlike the VRP or PDPTW problems, where all vehicles begin and end at a central depot, the ships in the TSS problems can be initially located anywhere and are not required to return to a central depot. Additionally, the size of the fleet is a given.

We listed a number of real ships in Table 6.2. For simplicity, we only choose from these possible ships to generate our fleets. The ships are chosen randomly, and so are their initial locations. Note that we use the same fleet for all test cases of the same size class. We also generated 4 more fleets for the small test cases to analyze the results on different fleets. The fleets are listed in Table 6.3.

*The paired-observation t test*

In order to compare the results of both models, and draw conclusions that are statistically valid, we make use of the paired-observation t-test. The chances of type II error is less in a paired-difference test than in other tests.[5] The reason is that pairing gets at the difference between two populations more directly. If the data is paired in some way, the difference between the two conveys more information about the difference between the two models, which means the evidence for hypotheses testing is stronger[1,

[5] Type II error in statistical hypotheses testing means to fail the rejection of a null hypothesis when it in fact is not true.

**Fleets for our experiments**

| Ship Nr | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| small | 1 | - | - | 1 | 2 |
| medium | 3 | 1 | 2 | 1 | 1 |
| large | 3 | 1 | 4 | 4 | 3 |
| small 2 | - | 1 | 1 | 1 | 1 |
| small 3 | 1 | - | 2 | - | 1 |
| small 4 | - | - | 2 | - | 2 |
| small 5 | 1 | - | 1 | 1 | 1 |

*Tab. 6.3:* Randomly generated fleets for our experiments. This table shows how each fleet is made up. The upper three fleets are used in all the main experiments, whereas the lower 4 fleets are only used for the experiments in Section 6.3.3.

Ch. 8-2]. Our data is paired, because the models are tested on the same test cases using the same fleets and the same random seeds when generating initial solutions[6]. This way, we remove extraneous variation that could have influenced the results. For instance, due to randomness, one model might start off with a superb initial solution. This could lead to wrong conclusions due to the randomness. Therefore, we keep the comparison as fair as possible by using the same random seeds for both models. Besides randomness in creating initial solutions, or actually insertion sequences, the TTS algorithm remains deterministic.

The test statistic for the paired-observation $t$-test in [1, Ch. 8.2] is given by:

$$t = \frac{\bar{D} - \mu_{D_0}}{s_D/\sqrt{n}} \tag{6.1}$$

where $\bar{D}$ is the average difference between the two populations, $s_D$ is the standard deviation of the differences, $n$ is the number of test observations (i.e. runs), and $\mu_{D_0}$ is the mean difference under the null hypothesis (which is 0 in our case). The statistic has a $t$ distribution with $n - 1$ degrees of freedom.

### *6.3.2 Computational results*

The profit values that we report here are calculated using Equation (3.17)[7], which subtracts the fuel costs from the revenues. Note that we deliberately excluded fixed costs, such as capital and maintenance costs in this equation. For one, these are difficult to obtain and quantify; should we calculate the capital costs of the whole fleet over the whole planning horizon or only until the last order of each ship? What is the life time of a ship, what are then the yearly costs of the investment? Should we regard the crew as a fixed cost as they will remain on board even though the ship may be idle and floating on sea? These questions are problematic. Also, incorporating these fixed costs have no direct impact on the model's solution. The optimal schedule that the model finds will remain the same; the according profit value will just be lowered

[6] The random seed determines the insertion sequence (See Section 5.1) for generating the initial solution. This however does not mean that the actual initial schedule is the same, since this is dependant on which model you use.

[7] When we use the TSS model without speed decisions, we can still use this equation, as the speed factor would simply be 1.

with a constant. For these reasons, we decided to simply neglect the fixed costs all together.

Note also that we neglect fixed port costs. If we would include these costs explicitly, it would have a direct impact on the revenue per order. However, since this value is significantly small relative to the revenues of these orders, we could say that they are implicitly taken into account. We randomize our revenue rate in our test case generator, so if the revenue rate is randomized to be $x$, we could say the actual revenue rate is slightly higher, but is lowered with the fixed port costs that is required pickup and deliver the respective order.

And since we do not incorporate these fixed costs that do occur in reality, we should keep in mind that the profits returned by the model are not the actual profits realized. However, this does not mean that the values are useless. We can still quantify the increasing profits of the TSS-SD model relative to the TSS model, as the fixed costs for both models are the same. This means that the increase in profits are an actual absolute increase in real profits.

This section presents the quantitative comparison between the TSS model and the TSS-SD model. We run both models on all of the 72 test cases, using 100 runs per experiment. We report the best profits, the mean profits, the standard deviation and the total CPU time in seconds. The results of the models are compared using a paired-difference test to draw statistically valid conclusions.

We compared the models on numerous test cases that differ in size and planning horizon. This gives us additional insights in the performance of the models on different types of cases. For example, it provides evidence to test our hypothesis that a wider planning horizon would imply more slow steaming opportunities, hence more advantage for the TSS-SD model. Also, we would like to quantify the additional computational efforts of the TSS-SD model relative to the TSS model on larger test cases as compared to smaller test cases.

For these experiments we used a fixed tenure of 30, as we opt for solution quality. We do not use the swap move, but we do deploy elite candidate lists. For the TSS-SD model, we choose a $\delta_s$ of 0.1, but we also performed the experiments using a $\delta_s$ of 0.01. The latter experiments would require more computational effort, but the optimal solution will be better. We would like to quantify this trade off.

*TSS-SD dominates TSS*

The results of the small test cases are summarized in Table 6.4. What we see at first glance is that the TSS-SD clearly dominates the TSS when it comes to profits. When we look at Table 6.4, and compare the results of the TSS model with the TSS-SD ($\delta_s = 0.1$) model, we see that the TSS-SD model outperforms the TSS model on all test cases with an extremely high significance at an alpha of 0.01% (with only one exception at an alpha of 1%, which is still highly significant). On average, the TSS-SD ($\delta_s = 0.1$) model achieves \$7.86 million savings on small test cases (i.e. 20 orders with 4 ships). The TSS-SD model, however, requires almost 7 times more computational effort. In absolute terms, this is only 1 second more on average per run, hence very insignificant given the tremendous savings potential.

The experimental results of the medium sized test cases (with 8 ships and 40 orders) are reported in Table 6.5. Likewise, with only a few exceptions, the TSS-SD ($\delta_s = 0.1$) model dominates the TSS model. The savings that are realized on medium sized test cases are higher, and amount on average \$11.5 million. The computational efforts are also about 7 times higher, but in this case this is on average about 11

seconds per run. Once again, very insignificant.

In Table 6.6 we report the results for the large test cases, which have a fleet of 15 ships and 100 orders. On all test cases, the TSS-SD model outperforms the TSS model with extremely high significance. On average, the savings amount $17.8 million. It requires 3 times as much computational effort, which is 14 seconds more per run in absolute terms.

The analysis of the results are summarized in Table 6.7. The bigger the test cases, the higher the potential savings from slow steaming. Additional computational effort increases too, but remains overshadowed by the potential savings. For instance, for large test cases, the TSS-SD model requires 21 seconds per run. In 10 minutes, the TSS-SD model is still able to create 30 solutions[8] from which the best schedule may be implemented.

*Planning horizons, wider is better.* Obviously, when the planning horizons become wider, i.e. the orders are more spread out over a wider planning horizon and with wider time windows, the profits rise. What we also see is that the difference between the TSS-model and the TSS model becomes bigger, which implies more slow steaming opportunity. Complexity wise, we do not see a significant increase for the TSS model as the planning horizon becomes larger. This is a different story for the TSS-SD model.

If we look at the small test cases with tight time windows (i.e. 20 orders with 4 ships in 40 days) in Table 6.4, we see that the TSS-SD saves on average $4.58 millions when compared to the TSS model. This compares to $7.64 millions when we have a planning horizon of 100 days and $11.35 millions on a planning horizon of 200 days. The increase in the additional computational effort can also be seen as the planning horizon becomes wider. The tight test cases require 2.75 times more computational efforts, which is about 0.3 seconds per run extra. For the normal test cases however, the TSS ($\delta_s = 0.1$) model needs over 5.5 times as much time, which is about 0.8 seconds extra per run in absolute terms. For wide planning horizons, this becomes 2.4 seconds per run, which is over 10 times as much when compared to the TSS model.

Similar proportions can also be observed for the medium sized test cases. Savings are on average $4.57 million, $12.25 million and $17.57 million for tight, normal and wide time windows respectively. Increase in computational efforts are on average 3.4 s/run (3.4x), 13.1 s/run (8.4x) and 16.7 s/run (10.8x) respectively.

For large test cases, the savings are on average $9.9 million, $17.8 million and $25.8 million respectively. The TSS-SD ($\delta_s = 0.1$) model requires on average 2.9 more seconds per run (1.4x), compared to 7.8 seconds (2.2x) for normal time windows and 31.5 seconds (5.4x) for wide time windows.

The wider the planning horizon, the more computation time is required for the TSS-SD model, but the more there is to be gained from slow steaming.

*A smaller step size is unnecessary* The $\delta_s$ denotes the step size of the speed heuristic as was described in Section 5.5.3. As we showed in the test case described in Section 6.2.3, lowering the step size allows the model to find a better optimum. Not only may this result in better optimal speeds on the legs, the resulting schedule might be different. On all test cases, we ran the model using a $\delta_s$ of 0.01 instead of 0.1. The results are reported in the right columns of Tables 6.4, 6.5 and 6.6.

At first glance, we see that the model with a $\delta_s$ of 0.01 does not perform significantly better than the model with a $\delta_s$ of 0.1. In fact, sometimes it returns a worse solution.

[8] Recall Figure 6.1 which shows that after 30 runs, the slope decreases quickly.

The computational time also increases dramatically. It is therefore not advised to opt for an even smaller step size. A $\delta_s$ of 0.1 is small enough, and remains very fast.

### 6.3.3 Further analysis

This section analysis two more aspects that are related to the model's outcome, i.e. using different tenure settings and using different fleets. These test are conducted only on the small test cases using a $\delta_s = 0.1$ for the TSS-SD model.

#### *Tenure testing*

How does the tenure values affect the performance of both the TSS and the TSS-SD model, both in solution quality and CPU time, and how does the relative difference change? We identified three dominating tabu tenure settings in Section 5.3, i.e. a fixed tenure of 30, 15 and the reactive tabu search with the parameters $x = 0.9$ and $y = 1$. The experiments are run on the small test case data. To make a fair comparison, the same initial solutions were used for every setting. The average results are summarized in Table 6.8. We also performed experiments using even higher tenure values, namely 45 and 60.

We see that the reactive tabu search is fastest as expected, with the lowest solution quality. The higher the fixed tenure value, the better the solution quality of both the TSS and the TSS-SD model, and the higher the required CPU time. We also see an increasing difference between the TSS and TSS-SD model. It seems that as the complexity of the algorithm increases, resulting in more computational effort but a higher optimum, the potential savings for slow steaming increases as well.

From a practical point of view, the user may opt to alter this parameter to find the right balance between computation time and solution quality, in addition to the parameter of number of runs. When there is ample time, the user would like to maximize his profits, and choose for a higher fixed tenure value in combination with a higher number of runs. However, we see that the improvement decreases quickly above a fixed tenure of 30, hence we advise using 30 for a good balance.

#### *Different fleets*

The randomly initialized fleets are separated from the randomly generated test cases. For the previous test cases, we used only three different fleets; one for small, one for medium and one for large. These fleets are listed in Table 6.3. To make sure the results are not fleet specific, we further analyze the small test cases using four more randomly initialized fleets, which are listed in the lower part of Table 6.3. When we randomized the fleets, we required the fleet to consist of at least one ship with a DWT capacity of 150000, since otherwise situations occur that orders are contracted but unable to be carried by the fleet. The results are presented in Table 6.9.

Clearly, the TSS-SD outperforms the TSS model no matter what fleet is being deployed. Another observation is that the average profits correlate positively with the total capacity of the fleet. In line with the tabu tenure experiments in the previous section, the absolute gap between the TSS-SD and the TSS model increases, when the proportion of profits increases. For instance, the average savings that the TSS-SD model realizes relative to the TSS model is almost \$6 million for fleet number 3, which made an average profit of \$3.67 million without speed optimization. When this latter increases to almost \$9 million for fleet number 1, the average increased savings increases to almost \$8 million.

**Small test cases**

| Data set | TSS model | | | | TSS-SD model ($\delta_s = 0.1$) | | | | TSS-SD model ($\delta_s = 0.01$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] |
| st1 | 16.84 | 8.13 | 5.70 | 17.11 | 21.76 | 15.06[c] | 4.87 | 43.75 | 21.77 | 15.10 | 4.84 | 304.79 |
| st2 | 15.14 | 4.06 | 7.23 | 16.66 | 18.49 | 6.83[d] | 7.84 | 53.26 | 18.48 | 6.73 | 8.13 | 364.04 |
| st3 | 4.44 | -7.56 | 4.97 | 14.12 | 8.41 | -4.18[c] | 5.10 | 41.93 | 8.43 | -4.14 | 5.23 | 294.65 |
| st4 | 20.50 | 10.07 | 6.69 | 17.85 | 25.69 | 15.43[c] | 6.78 | 52.33 | 25.69 | 15.14 | 6.80 | 361.64 |
| st5 | 15.56 | 4.27 | 7.21 | 13.54 | 22.37 | 10.92[c] | 7.89 | 35.43 | 22.38 | 11.33 | 7.88 | 243.52 |
| st6 | 9.31 | -3.36 | 5.22 | 14.02 | 11.99 | -0.34[c] | 4.74 | 35.55 | 11.99 | -0.33 | 4.77 | 246.71 |
| st7 | 2.81 | -10.83 | 5.13 | 15.96 | 3.68 | -8.04[c] | 4.34 | 35.80 | 3.68 | -8.01 | 4.40 | 235.74 |
| st8 | 17.24 | 1.97 | 5.73 | 15.10 | 18.42 | 7.70 [c] | 4.76 | 44.54 | 18.43 | 7.87 | 4.89 | 308.26 |
| **Avg st** | **12.73** | **0.85** | **5.99** | **15.54** | **16.35** | **5.42** | **5.79** | **42.82** | **16.36** | **5.46** | **5.87** | **294.92** |
| sn1 | 23.59 | 9.81 | 7.70 | 19.13 | 30.24 | 18.94[c] | 6.21 | 95.59 | 30.63 | 18.91 | 6.18 | 780.90 |
| sn2 | 28.63 | 14.84 | 6.40 | 18.75 | 35.38 | 23.33[c] | 5.11 | 111.03 | 36.10 | 23.07 | 4.93 | 897.27 |
| sn3 | 12.87 | -5.38 | 8.04 | 13.26 | 23.60 | 1.38[c] | 7.84 | 46.90 | 23.44 | 1.25 | 7.85 | 341.76 |
| sn4 | 18.16 | 8.56 | 4.68 | 12.61 | 24.01 | 15.17[c] | 4.53 | 63.12 | 24.04 | 15.14 | 4.56 | 503.81 |
| sn5 | 29.01 | 18.61 | 7.42 | 24.68 | 37.26 | 28.07[c] | 3.51 | 137.52 | 37.27 | 27.97 | 3.44 | 1130.57 |
| sn6 | 27.22 | 19.46 | 3.08 | 22.23 | 32.44 | 27.33[c] | 2.72 | 130.50 | 32.45 | 27.69 | 2.54 | 1141.13 |
| sn7 | 20.25 | 12.06 | 5.01 | 16.95 | 27.55 | 18.38[c] | 4.92 | 80.02 | 26.14 | 18.39 | 4.97 | 605.85 |
| sn8 | 28.15 | 14.17 | 4.83 | 18.49 | 35.20 | 20.65[c] | 4.66 | 144.98 | 35.22 | 20.78 | 4.29 | 1223.53 |
| **Avg sn** | **23.48** | **11.52** | **5.90** | **18.26** | **30.71** | **19.16** | **4.94** | **101.21** | **30.66** | **19.15** | **4.84** | **828.10** |
| sw1 | 27.64 | 21.67 | 3.51 | 33.56 | 38.44 | 33.08[c] | 2.89 | 261.37 | 38.90 | 32.87 | 2.88 | 2233.46 |
| sw2 | 22.36 | 14.59 | 4.68 | 26.83 | 32.01 | 24.33[c] | 4.86 | 304.79 | 32.02 | 24.02 | 5.03 | 2702.81 |
| sw3 | 30.21 | 26.09 | 2.14 | 33.75 | 42.64 | 36.67[c] | 3.10 | 220.60 | 42.68 | 36.97 | 2.98 | 1923.33 |
| sw4 | 24.01 | 5.49 | 6.97 | 22.91 | 26.17 | 16.49[c] | 4.78 | 341.10 | 25.26 | 16.49 | 4.66 | 3284.97 |
| sw5 | 14.46 | 1.55 | 6.54 | 22.85 | 23.61 | 14.37[c] | 6.53 | 214.20 | 23.63 | 14.18 | 6.48 | 1788.10 |
| sw6 | 29.16 | 23.80 | 3.47 | 33.65 | 38.97 | 32.69[c] | 4.28 | 405.70 | 38.97 | 32.70 | 4.25 | 3525.17 |
| sw7 | 27.42 | 11.68 | 9.91 | 19.00 | 37.04 | 24.64[c] | 7.86 | 181.38 | 37.05 | 24.56 | 7.85 | 1508.66 |
| sw8 | 23.64 | 11.80 | 7.83 | 16.97 | 36.98 | 25.17[c] | 5.76 | 220.19 | 36.50 | 24.83 | 5.44 | 1901.87 |
| **Avg sw** | **24.86** | **14.58** | **5.63** | **26.19** | **34.48** | **25.93** | **5.01** | **268.67** | **34.38** | **25.83** | **4.95** | **2358.54** |
| **Total Avg** | **20.36** | **8.98** | **5.84** | **20.00** | **27.18** | **16.84** | **5.24** | **137.57** | **27.13** | **16.81** | **5.22** | **1160.52** |

[a] Profits are reported in millions.
[b] CPU time is given in seconds; run on Intel Core2 Duo, 2.1Ghz, 2GB Ram. Note that this is the total CPU time for 100 runs.
[c] Outperforms the TSS model with a statistical significance at an alpha of 0.01%.
[d] Outperforms the TSS model with a statistical significance at an alpha of 1%.

*Tab. 6.4:* Comparative computational results for 24 small test cases, using 4 ships and 20 orders. The values are obtained through 100 runs per test case. The TSS-SD ($\delta_s = 0.1$) model dominates the TSS model with an average saving of $7.86 million. Additional computation time relative to the TSS model is only slightly more than a second per run.

**Medium test cases**

| Data set | TSS model | | | | TSS-SD model ($\delta_s = 0.1$) | | | | TSS-SD model ($\delta_s = 0.01$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] |
| mt1 | 100.34 | 91.21 | 5.20 | 131.32 | 111.05 | 99.55[c] | 4.51 | 329.03 | 111.09 | 99.36 | 4.26 | 2174.52 |
| mt2 | 82.23 | 70.40 | 5.64 | 164.19 | 96.87 | 80.21[c] | 6.87 | 698.32 | 93.56 | 80.30 | 6.62 | 5215.94 |
| mt3 | 83.50 | 66.27 | 7.15 | 160.09 | 88.12 | 67.78[d] | 8.29 | 574.20 | 83.49 | 67.43 | 7.70 | 4414.17 |
| mt4 | 88.68 | 76.83 | 6.58 | 150.33 | 100.37 | 79.14[e] | 8.08 | 449.04 | 99.91 | 78.20 | 7.22 | 2918.19 |
| mt5 | 92.38 | 70.73 | 7.22 | 133.94 | 102.06 | 79.31[c] | 7.80 | 404.30 | 97.92 | 80.18 | 8.06 | 2601.79 |
| mt6 | 79.64 | 73.59 | 3.83 | 131.66 | 89.75 | 80.14[c] | 5.76 | 519.19 | 92.85 | 80.30 | 5.46 | 3992.23 |
| mt7 | 69.77 | 44.37 | 9.54 | 114.17 | 69.75 | 46.40[f] | 9.12 | 430.23 | 67.07 | 46.99 | 9.56 | 3063.83 |
| mt8 | 93.86 | 84.62 | 5.59 | 118.87 | 99.28 | 82.05[c] | 7.85 | 393.49 | 100.27 | 82.71 | 7.68 | 2574.84 |
| **Avg mt** | **86.30** | **72.25** | **6.34** | **138.07** | **94.66** | **76.82** | **7.29** | **474.72** | **93.27** | **76.93** | **7.07** | **3369.44** |
| mn1 | 78.63 | 69.84 | 6.72 | 209.38 | 99.09 | 77.58[c] | 7.64 | 2005.91 | 99.09 | 78.37 | 8.33 | 16895.54 |
| mn2 | 61.15 | 56.66 | 6.04 | 186.35 | 82.16 | 73.37[c] | 7.26 | 2177.84 | 82.17 | 73.36 | 7.27 | 18586.22 |
| mn3 | 84.25 | 77.82 | 5.73 | 191.41 | 104.61 | 88.84[c] | 7.69 | 1573.69 | 103.71 | 88.77 | 7.07 | 13679.11 |
| mn4 | 92.80 | 84.32 | 5.46 | 172.97 | 113.24 | 94.66[c] | 7.38 | 1191.57 | 111.31 | 96.07 | 8.04 | 11044.89 |
| mn5 | 70.68 | 66.77 | 3.13 | 177.53 | 90.26 | 81.53[c] | 6.55 | 1433.13 | 89.79 | 81.82 | 6.59 | 13180.46 |
| mn6 | 68.84 | 61.34 | 4.03 | 145.90 | 89.01 | 76.08[c] | 5.49 | 1060.70 | 88.49 | 75.39 | 5.48 | 9222.85 |
| mn7 | 87.14 | 81.95 | 4.06 | 201.07 | 106.86 | 92.68[c] | 7.05 | 1536.81 | 106.93 | 92.62 | 7.08 | 13888.88 |
| mn8 | 63.56 | 48.03 | 8.51 | 141.32 | 79.25 | 60.00[c] | 8.10 | 960.84 | 79.25 | 59.26 | 8.36 | 8006.55 |
| **Avg mn** | **75.88** | **68.34** | **5.46** | **178.24** | **95.56** | **80.59** | **7.14** | **1492.56** | **95.09** | **80.71** | **7.28** | **13063.06** |
| mw1 | 91.49 | 83.44 | 6.39 | 252.50 | 114.30 | 94.14 [c] | 10.44 | 3362.31 | 114.31 | 93.97 | 10.27 | 32770.21 |
| mw2 | 86.37 | 83.25 | 3.47 | 171.72 | 109.82 | 103.73[c] | 6.42 | 2458.96 | 109.76 | 103.37 | 6.45 | 22455.82 |
| mw3 | 89.99 | 64.60 | 14.17 | 110.98 | 117.55 | 101.50[c] | 10.10 | 1237.91 | 117.56 | 101.69 | 9.15 | 10915.09 |
| mw4 | 82.39 | 72.90 | 6.48 | 153.10 | 102.63 | 88.87[c] | 8.95 | 1379.01 | 107.69 | 89.61 | 8.91 | 12253.60 |
| mw5 | 94.54 | 55.48 | 15.94 | 129.26 | 106.19 | 76.12[c] | 13.15 | 1164.52 | 106.20 | 75.37 | 12.94 | 9926.32 |
| mw6 | 82.32 | 79.30 | 2.49 | 235.39 | 104.18 | 98.02[c] | 5.72 | 2651.18 | 104.16 | 97.91 | 5.96 | 23530.50 |
| mw7 | 68.88 | 55.77 | 8.78 | 168.72 | 83.31 | 67.25[c] | 7.57 | 1364.78 | 83.15 | 66.61 | 7.25 | 12227.34 |
| mw8 | 88.49 | 69.37 | 9.34 | 136.28 | 99.63 | 75.02[c] | 10.22 | 1104.40 | 99.64 | 75.07 | 9.89 | 9561.63 |
| **Avg mw** | **85.56** | **70.51** | **8.38** | **169.74** | **104.70** | **88.08** | **9.07** | **1840.38** | **105.31** | **87.95** | **8.85** | **16705.06** |
| **Total Avg** | **82.58** | **70.37** | **6.73** | **162.02** | **98.31** | **81.83** | **7.83** | **1269.22** | **97.89** | **81.86** | **7.73** | **11045.86** |

[a] Profits are reported in millions.
[b] CPU time is given in seconds; run on Intel Core2 Duo, 2.1Ghz, 2GB Ram. Note that this is the total CPU time for 100 runs.
[c] Outperforms the TSS model with a statistical significance at an alpha of 0.01%.
[d] Outperforms the TSS model with a statistical significance at an alpha of 10%.
[e] Outperforms the TSS model with a statistical significance at an alpha of 1%.
[f] Outperforms the TSS model with a statistical significance at an alpha of 5%.

*Tab. 6.5:* Comparative computational results for 24 medium sized test cases, using 8 ships and 40 orders. The values are obtained through 100 runs per test case. The TSS-SD ($\delta_s = 0.1$) model outperforms the TSS model significantly with only one exception, i.e. mt3 with a 10% alpha. For test case mt7 this is a 5% alpha significance and mt3 a 1% alpha significance. All other test cases are dominated by the TSS-SD with a 0.01% alpha.

**Large test cases**

| Data set | TSS model | | | | TSS-SD model ($\delta_s = 0.1$) | | | | TSS-SD model ($\delta_s = 0.01$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] | Best[a] | Mean[a] | Stdv[a] | CPU[b] |
| lt1 | -29.61 | -37.38 | 11.48 | 626.55 | 5.22 | -25.25[c] | 10.93 | 841.21 | 5.24 | -25.18 | 10.35 | 2549.81 |
| lt2 | 8.90 | -17.89 | 12.82 | 738.73 | 18.73 | -8.67[c] | 12.30 | 895.11 | 23.01 | -9.24 | 12.35 | 2841.93 |
| lt3 | 22.79 | -18.71 | 14.08 | 677.25 | 23.50 | -11.47[c] | 14.41 | 831.45 | 24.03 | -11.14 | 13.90 | 2710.84 |
| lt4 | 10.86 | -20.75 | 10.12 | 698.71 | 12.43 | -12.46[c] | 12.14 | 841.64 | 12.46 | -12.36 | 11.97 | 1983.78 |
| lt5 | -9.37 | -38.56 | 12.01 | 671.51 | -0.71 | -33.13[c] | 9.76 | 1096.38 | -0.72 | -32.36 | 10.19 | 4223.76 |
| lt6 | -24.20 | -33.24 | 12.10 | 594.12 | 3.10 | -19.13[c] | 13.10 | 1027.42 | 8.64 | -18.83 | 13.38 | 2128.00 |
| lt7 | 26.81 | -13.99 | 18.05 | 560.16 | 30.62 | -0.44[c] | 17.07 | 933.72 | 33.81 | 0.52 | 17.38 | 1964.47 |
| lt8 | -32.36 | -40.48 | 13.74 | 746.45 | -37.30 | -31.06[c] | 9.91 | 1195.99 | -40.05 | -30.83 | 10.38 | 2432.10 |
| **Avg lt** | **-3.27** | **-27.63** | **13.05** | **664.19** | **6.95** | **-17.70** | **12.45** | **957.86** | **8.30** | **-17.43** | **12.49** | **2604.34** |
| ln1 | 22.94 | -12.27 | 14.54 | 737.03 | 39.00 | 5.30[c] | 13.01 | 2393.39 | 40.28 | 4.90 | 14.33 | 11511.59 |
| ln2 | 43.07 | 2.21 | 16.02 | 679.70 | 55.32 | 19.79[c] | 14.16 | 1078.34 | 56.75 | 21.19 | 14.23 | 4105.35 |
| ln3 | 11.92 | -30.63 | 16.08 | 642.89 | 47.13 | -5.99[c] | 16.17 | 1291.61 | 36.17 | -4.74 | 16.05 | 6236.28 |
| ln4 | 10.88 | -28.71 | 16.27 | 678.18 | 24.69 | -16.40[c] | 16.32 | 1283.54 | 29.16 | -16.86 | 17.00 | 5708.13 |
| ln5 | 47.16 | 1.54 | 14.27 | 716.96 | 60.15 | 23.34[c] | 12.72 | 1731.22 | 57.87 | 24.30 | 12.80 | 9421.48 |
| ln6 | 25.58 | -15.31 | 16.01 | 657.88 | 29.01 | -2.48[c] | 14.78 | 1352.04 | 36.10 | -2.30 | 15.56 | 6645.00 |
| ln7 | 17.56 | -20.53 | 14.78 | 701.74 | 38.16 | -1.28[c] | 14.57 | 1302.70 | 32.78 | -1.18 | 14.47 | 5540.33 |
| ln8 | 22.88 | -12.31 | 15.67 | 633.68 | 52.75 | 4.34[c] | 15.74 | 1289.32 | 40.99 | 3.43 | 14.86 | 6064.68 |
| **Avg ln** | **25.25** | **-14.50** | **15.45** | **681.01** | **43.28** | **3.33** | **14.68** | **1465.27** | **41.26** | **3.59** | **14.91** | **6904.10** |
| lw1 | 39.32 | -14.32 | 18.34 | 626.38 | 50.72 | 11.69[c] | 15.95 | 3317.39 | 49.50 | 11.36 | 15.76 | 25093.24 |
| lw2 | 72.48 | 47.95 | 13.05 | 762.79 | 119.29 | 77.78[c] | 14.51 | 3415.72 | 117.63 | 77.31 | 14.95 | 24685.81 |
| lw3 | 18.53 | -13.15 | 12.20 | 790.92 | 44.35 | 7.11[c] | 14.05 | 4582.83 | 42.05 | 6.60 | 14.21 | 40673.85 |
| lw4 | 65.74 | 24.59 | 18.03 | 612.49 | 88.84 | 52.19[c] | 19.24 | 2624.35 | 92.56 | 52.37 | 19.27 | 19150.44 |
| lw5 | 60.93 | 7.36 | 12.89 | 801.81 | 67.09 | 34.61[c] | 12.43 | 5046.41 | 67.11 | 35.61 | 12.83 | 34868.89 |
| lw6 | 45.94 | 5.39 | 17.54 | 610.01 | 73.32 | 32.33[c] | 16.50 | 4146.93 | 73.37 | 32.09 | 17.25 | 20768.83 |
| lw7 | 64.68 | 24.91 | 13.71 | 831.52 | 86.15 | 43.99[c] | 14.07 | 4578.46 | 82.94 | 43.78 | 13.29 | 32360.09 |
| lw8 | 31.18 | -2.94 | 14.23 | 726.15 | 59.17 | 26.09[c] | 14.28 | 3283.62 | 64.04 | 26.18 | 13.72 | 21367.28 |
| **Avg lw** | **49.85** | **9.97** | **15.00** | **720.26** | **73.62** | **35.72** | **15.13** | **3874.46** | **73.65** | **35.66** | **15.16** | **27371.05** |
| **Total Avg** | **23.94** | **-10.72** | **14.50** | **688.48** | **41.28** | **7.12** | **14.09** | **2099.20** | **41.07** | **7.28** | **14.19** | **12293.16** |

[a] Profits are reported in millions.
[b] CPU time is given in seconds; run on Intel Core2 Duo, 2.1Ghz, 2GB Ram. Note that this is the total CPU time for 100 runs.
[c] Outperforms the TSS model with a statistical significance at an alpha of 0.01%.

*Tab. 6.6:* Comparative computational results for 24 large test cases, using 15 ships and 100 orders. The values are obtained through 100 runs per test case. On all test cases, the TSS-SD model outperforms the TSS model with an enormous significance.

**Summary analysis of experimental results**

| Test cases | | Profits[a] | CPU[b] | (factor) |
|---|---|---|---|---|
| small | tight | 4.58 | 27.28 | (2.75) |
| | normal | 7.64 | 82.94 | (5.54) |
| | wide | 11.35 | 242.48 | (10.26) |
| | Average | 7.86 | 117.57 | (6.88) |
| medium | tight | 4.57 | 336.65 | (3.44) |
| | normal | 12.25 | 1314.32 | (8.37) |
| | wide | 17.57 | 1670.64 | (10.84) |
| | Average | 11.46 | 1107.20 | (7.83) |
| large | tight | 9.92 | 293.68 | (1.44) |
| | normal | 17.83 | 784.26 | (2.15) |
| | wide | 25.75 | 3154.21 | (5.38) |
| | Average | 17.84 | 1410.72 | (3.05) |

*Tab. 6.7:* Results reported in this table report the average differences between the TSS-SD ($\delta_s = 0.1$) model and the TSS model. Values are derived from Tables 6.4, 6.5 and 6.6.

[a] Profits are reported in millions.
[b] CPU time is given in seconds; run on Intel Core2 Duo, 2.1Ghz, 2GB Ram. Note that this is the total CPU time for 100 runs.

**Tenure testing**

| Tabu Tenure | **TSS model** | | **TSS-SD model** | | Difference | |
|---|---|---|---|---|---|---|
| | Avg[a] | CPU[b] | Avg[a] | CPU[b] | Avg[a] | CPU[b](factor) |
| RTS[c] | 5.06 | 6.12 | 11.44 | 44.17 | 6.37 | 38.05(7.21) |
| fixed 15 | 7.22 | 16.24 | 14.27 | 102.11 | 7.05 | 85.87(6.29) |
| fixed 30 | 8.98 | 20.00 | 16.84 | 137.57 | 7.86 | 117.57(6.88) |
| fixed 45 | 9.41 | 23.06 | 17.42 | 209.81 | 8.01 | 186.75(9.10) |
| fixed 60 | 9.37 | 26.88 | 17.44 | 246.16 | 8.07 | 219.29(9.16) |

[a] Average profits are reported in millions.
[b] CPU time is given in seconds for 100 runs.
[c] Reactive tabu search with the parameters $x = 0.9$ and $y = 1$.

*Tab. 6.8:* Computational results of the experiments on the small test case data using different tabu tenure settings.

**Different fleets**

| Fleet nr | **TSS model** | | **TSS-SD model** | | Difference | |
|---|---|---|---|---|---|---|
| | Avg[a] | CPU[b] | Avg[a] | CPU[b] | Avg[a] | CPU[b](factor) |
| 1 | 8.98 | 20.00 | 16.84 | 137.57 | 7.86 | 117.57(6.88) |
| 2 | 4.22 | 17.07 | 10.51 | 106.20 | 6.29 | 89.13(6.22) |
| 3 | 3.67 | 13.26 | 9.64 | 102.68 | 5.96 | 89.43(7.75) |
| 4 | 8.54 | 15.61 | 16.47 | 140.75 | 7.93 | 125.14(9.02) |
| 5 | 4.46 | 13.75 | 10.96 | 104.42 | 6.50 | 90.67(7.59) |

[a] Average profits are reported in millions.
[b] CPU time is given in seconds for 100 runs.

*Tab. 6.9:* Different fleets yield different results. More profits are to be achieved when the total capacity of the fleet increases. The TSS-SD model outperforms the TSS model nonetheless on every case.

### *6.3.4 Summary*

A thorough description of the experimental methodology and the corresponding results of our extensive quantitative comparisons are discussed in this chapter.

We developed our models by first calibrating the performance of our general TSS model with the well studied PDPTW model on benchmark instances. This benchmark study is described in Section 6.1. We made sure that the TSS model was performing well enough for our purpose before we expanded it into the TSS-SD model.

This chapter also presented a number of test cases to verify and validate our models, presented in Section 6.2. Three small cases are presented to demonstrate the potential of the TSS-SD model.

Section 6.3 presents the extensive comparison between the two models on generated ship scheduling instances. We described how we chose the parameters to our test case generator in an attempt to match reality as closely as possible. This is important since otherwise the resulting savings will not be representative at all if the input values are out of proportion.

On only one out of 72 test cases the TSS-SD did not significantly outperform the TSS model with a 5% significance, two with a 1% significance, and only four with a 0.01% significance. On the remaining 68 test cases, the TSS-SD model dominates the TSS model with great significance.

# 7. CONCLUSION AND DISCUSSION

Two distinct models were developed to quantify the consequences of incorporating cruising speed decisions in the tramp ship scheduling problem. One model does not optimize cruising speeds and optimizes the schedules using the conventional way of assigning a fixed speed. The other model simultaneously optimizes the sailing routes and the cruising speeds on the legs, with an accuracy to a tenth of a knot. Both models were compared on solution quality and computational effort, on numerous test cases that were generated to match reality. Our new model clearly dominates the conventional model; by incorporating speed decisions into the model may result in multi million dollar savings in fuel costs, which is the main driver of operating costs, without major additional computational effort.

These models apply customized tabu search algorithms for quick and good solutions, which makes the tool practical for the dynamic tramp industry. In practice, the tool enables the user to quickly generate new schedules whenever a new order occurs on the spot market. Speed is of essence, but when additional millions of dollars of savings are potential, the user may opt to continue running until a decision has to be made. This gives the user flexibility to come up with really good solutions in just very little time, or allow it to keep running for as long as his situation permits, to maximize his profits. All parameters, such as fuel price or current locations or loads of the ships, can be automatically updated to real time values.

The additional time that is required for the extended model is very insignificant compared to the potential savings. This is due to the highly efficient heuristic that we developed and embedded into the tabu search, making no significant compromises in the search space of the tabu search.

As noted in Section 6.3.2, the values that we obtained are not actual profits realized, since we neglect other fixed costs that occur in reality. The values are calculated using the objective functions described in Chapter 3, and technically should not be called profits, but rather fitness values. Care should be taken in practice, as a profitable schedule returned by the model might not be profitable in reality. Nonetheless, the model still succeeds in minimizing the losses in that case.

Furthermore, since this area of research is fairly new, no benchmark instances has yet been developed in literature. We generated our own benchmark instances for our research, in which we attempt to simulate a wide range of different real cases as closely as possible. The test cases that we generated may deviate from real life cases however[1], which may result in a different proportion of savings when compared to those reported in this study. For future research it would be really interesting to span the bridge between academics and practice, and to study numerous real life cases with real shipping companies.

Also we would like to point out a number of aspects that need to be kept in mind. Firstly, the models that we present in this study are rather on an abstract macro

[1] which may also be due to the specifics of a company, changes in the economy, etc.

level, hence neglecting details such as weather influences, ocean currents, pirates, and other real life factors that eventually influence the bottom line. Also, we assumed that the minimum speed of all ships is half the maximum speed. This is chosen for simplicity, but may be specified manually per ship for more accuracy. Finally, and most importantly, the results of this study assumes the accuracy of the simplified model of [27] to quantify the relationship between fuel consumption and speed. In order to make the model more accurate, explicit experimenting needs to be conducted with the real respective ships of the fleet in order to gather real data per ship and it's respective slow steaming potential. This would enable us to not use the estimation model anymore.

Nonetheless, it is still safe to say that implementing our TSS-SD model is better than our TSS model. We showed the potential and practicality of incorporating speed decisions into scheduling problems. It is also safe to say that by doing so, a tramp shipping company, depending on their fleet size and number of orders, may save tens of millions of dollars a year in operating expenses while magnificently reducing the carbon footprint. Once again, we would really like to stimulate future research on real cases, using real data instead of simulated and approximated data to validate our research.

Another direction for future research is to implement and study the TSS-SDSTW model that we described in Section 3.4. Soft time windows allow for more slow steaming flexibilities, but makes the problem at hand even more complex.

Note also that for our experiments, we compared the TSS model with the TSS-SD models by comparing the results with one another on a fixed number of 100 runs. The reason for this methodology is that we able to compare the two models while keeping the element of randomness to the minimum. It also enables us to perform a statistically strong paired observation $t$ test comparison.

For shipping companies, it would also be interesting to know how the two models perform, given a certain time available. The TSS-SD model requires more time per run, but is able to achieve high savings per run. The TSS model is faster and may create more solutions in the same amount of time, from which the best schedule is selected. Hypothetically, when very little time is available, and the TSS-SD model may only generate a few solutions, the TSS model might perform better. Even then, using speed optimization afterwards to improve a given schedule may even result in great savings.

# BIBLIOGRAPHY


[1] A. D. Aczel. *Complete Business Statistics*. McGraw-Hill, 2002.

[2] L. H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3(1):53–68, 1969.

[3] D. Bausch, G. Brown, and D. Ronen. Scheduling short-term marine transport of bulk products. *Maritime Policy and Management*, 25(4):335–348, 1998.

[4] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: aclassification scheme and survey. *TOP*, 15:1–31, 2007.

[5] G. Brønmo, M. Christiansen, K. Fagerholt, and B. Nygreen. A multi-start local search heuristic for ship scheduling–a computational study. *Computers & Operations Research*, 34(3):900 – 917, 2007.

[6] G. Brønmo, M. Christiansen, and B. Nygreen. Ship routing and scheduling with flexible cargo sizes. *The Journal of the Operational Research Society*, 58(9):1167, September 2007.

[7] G. Brønmo, B. Nygreen, and J. Lysgaard. Column generation approaches to ship scheduling with flexible cargo sizes. *European Journal of Operational Research*, In Press, Corrected Proof:–, 2009. doi: 10.1016/j.ejor.2008.12.028.

[8] G. G. Brown, G. W. Graves, and D. Ronen. Scheduling ocean transportation of crude oil. *Management Science*, 33(3):335–346, 1987.

[9] M. Christiansen, K. Fagerholt, and D. Ronen. Ship Routing and Scheduling: Status and Perspectives. *TRANSPORTATION SCIENCE*, 38(1):1–18, 2004.

[10] M. Christiansen, K. Fagerholt, B. Nygreen, and D. Ronen. Chapter 4 maritime transportation. In C. Barnhart and G. Laporte, editors, *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 189–284. Elsevier, 2007.

[11] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, 1979.

[12] J.-F. Cordeau and G. Laporte. Tabu search heuristics for the vehicle routing problem. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization Via Memory and Evolution*. Kluwer Academic Publishers, 2005.

[13] K. Fagerholt. Ship scheduling with soft time windows: An optimisation based approach. *European Journal of Operational Research*, 131(3):559 – 571, 2001.

[14] K. Fagerholt. A computer-based decision support system for vessel fleet scheduling–experience and future research. *Decision Support Systems*, 37(1):35–47, 2004.

[15] R. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Bell Laboratories, Murray Hill, NJ, 1979.

[16] M. Gendreau. An introduction to tabu search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[17] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[18] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40:1276–1290, 1994.

[19] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533 – 549, 1986.

[20] F. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

[21] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[22] S.-H. Kim and K.-K. Lee. An optimization-based decision support system for ship scheduling. *Computers & Industrial Engineering*, 33(3-4):689–692, 1997. Selected Papers from the Proceedings of 1996 ICC.

[23] J. E. Korsvik and K. Fagerholt. A tabu search heuristic for ship routing and scheduling with flexible cargo quantities. *Journal of Heuristics*, 2008.

[24] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4-5):285–300, 2000.

[25] S. A. Lawrence. *International Sea Transport: The Years Ahead*. Lexington Books, Lexington, MA, 1972.

[26] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pages 160–167, 2001.

[27] G. C. Manning. *The Theory and Technique of Ship Design*. MIT Press, Cambridge, Massachusetts, 1956.

[28] W. P. Nanry and J. W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34(2): 107–121, 2000.

[29] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems. *Journal für betriebswirtschaft*, 58:81–117, 2008.

[30] Y. Rochat and ric D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

[31] D. Ronen. The effect of oil price on the optimal speed of ships. *The Journal of the Operational Research Society*, 33(11):1035–1040, 1982. ISSN 01605682.

[32] D. Ronen. Cargo ships routing and scheduling: Survey of models and problems. *European Journal of Operational Research*, 12(2):119–126, February 1983.

[33] D. Ronen. Ship scheduling: The last decade. *European Journal of Operational Research*, 71(3):325–333, 1993.

[34] P. Toth and D. Vigo. The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS JOURNAL ON COMPUTING*, 15(4):333–346, 2003.

[35] UNCTAD. *Review of Maritime Transport*. United Nations, New York and Geneva, 2008.