A Heuristic and Hybrid Method for the Tank Allocation Problem in Maritime Bulk Shipping

Vilhelmsen, Charlotte; Larsen, Jesper; Lusby, Richard Martin

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

A Heuristic and Hybrid Method for the Tank Allocation Problem in Maritime Bulk Shipping

Charlotte Vilhelmsen Jesper Larsen Richard M. Lusby

Department of Management Engineering, Technical University of Denmark
Produktionstorvet, Building 426, 2800 Kgs. Lyngby, Denmark
chaan@dtu.dk, jesla@dtu.dk, rmlu@dtu.dk

July 31, 2014

Abstract

In bulk shipping, ships often have multiple tanks and carry multiple inhomogeneous products at a time. When operating such ships it is therefore a major challenge to decide how to best allocate cargoes to available tanks while taking into account tank capacity, safety restrictions, ship stability and strength as well as other operational constraints. The problem of finding a feasible solution to this tank allocation problem has been shown to be *NP-Complete*. We approach the problem on a tactical level where requirements for computation time are strict while solution quality is less important than simply finding a feasible solution. We have developed a heuristic that can efficiently find feasible cargo allocations. Computational results show that it can solve 99% of the considered instances within 0.4 seconds and all of them if allowed longer time. We have also modified an optimality based method from the literature. The heuristic is much faster than this modified method on the vast majority of considered instances. However, the heuristic struggles on two instances which are relatively quickly solved by the modified optimality based method. These two methods therefore complement each other nicely and so, we have created a hybrid method that first runs the heuristic and if the heuristic fails to solve the problem, then runs the modified optimality based method on the parts of the problem that the heuristic did not solve. This hybrid method cuts between 90% and 94% of the average running times compared to the other methods and consistently solves more instances than the other methods within any given time limit. In fact, this hybrid method is fast enough to be used in a tactical setting.

1 Introduction

Every year 8.7 billion tons of goods or equivalently 80% of world trade by volume is carried by ships (UNCTAD, 2012). This translates into well over a tonne of cargo for every single individual on the planet, every single year, and the global economy therefore depends on the international shipping industry's efficiency and competitive freight rates. Hence, research to increase efficiency within maritime transportation is important, and, taking the mere size of this huge industry into consideration, even small improvements can have great impact.

In this paper we consider bulk shipping, both wet and dry. Many bulk ships have multiple tanks and can thereby carry multiple inhomogeneous products at a time. Two examples of such ships are oil product tankers and chemical tankers. A major challenge when operating such ships is how to best allocate cargoes to available tanks while taking tank capacity, safety restrictions for onboard cargoes, ship stability and strength as well as other operational constraints into account. The complexity of the allocation problem varies with the number of tanks and the number and type of different products transported simultaneously. A chemical tanker can for instance have as many as 50 different tanks and hazmat (hazardous materials) regulations play a major role when allocating the products to the different tanks. E.g. products in neighboring tanks must

be non-reactive and incompatible products must not succeed each other in a tank unless it is cleaned (this can be costly). The regulations on product succession mean that we need to keep track of previous tank allocations and that decisions at any voyage leg affect decisions at future voyage legs complicating the problem even further. Often it is not allowed to move a cargo once it has been allocated to tanks and then this interdependency between voyage legs becomes even stronger. Taking stability, safety restrictions and other operational constraints into consideration it can therefore be extremely difficult, if not impossible, to find a feasible allocation for a given set of cargoes. In fact, Hvattum et al. (2009) show that the problem of finding a feasible solution is *NP-Complete*.

The *Tank Allocation Problem* (TAP) as described above is an operational planning problem and so, it is normally solved for a given route, i.e. *after* the fleet routes and schedules have been created. However, separating these two planning problems means that we can potentially create routes and schedules for which no feasible tank allocation exists. Therefore, for bulk fleets the tank allocation aspect should be included in the routing and scheduling phase of planning. Tank cleaning costs and other costs related to tank allocations are insignificant compared to the profits from carrying cargoes. Hence, at the tactical planning level where routes and schedules are determined, we can simplify the tank allocation aspect by ignoring the allocations costs and simply focus on finding a feasible allocation. Note that this also means that we can refrain from keeping track of past cargo allocations since we can just assume that all tanks are cleaned.

Within shipping and many other areas, routing and scheduling problems are often solved in a way that requires assessment of numerous routes, as for instance in column generation and local search based methods. For each considered route, the TAP must be solved to assess route feasibility with respect to stowage. The solution time for the entire procedure will therefore only be acceptable if the TAP can be solved efficiently. Furthermore, uncertainty plays a big part in maritime optimization where planners face a constantly changing environment with large daily variations in demand and many unforeseen events and so, it is often necessary to re-plan routes and schedules continuously to accommodate new cargoes and changes to existing plans. In effect, this means that the TAP must be solved repeatedly and that the requirements for computation time are strict.

Bulk operators of large and even medium size can easily have fleets with more than 25 ships. Since these ships carry multiple cargoes onboard simultaneously, the combinatorial puzzle associated with routing the ships is much larger than for ships sailing full shiploads, i.e. just one cargo onboard at a time. Thereby, there can be easily be more than 200 routes to evaluate for each ship and so, it is necessary to assess feasibility with respect to stowage for at least $25 \cdot 200 = 5000$ routes. If we allow a run time of up to just 0.25 seconds, assessing feasibility for these routes can alone take $0.25 \cdot 5000 = 1250$ seconds, i.e. 21 minutes. Hence, if we want to develop a method that is applicable to bulk operators of all sizes, the requirements for computation are quite strict.

Hvattum et al. (2009) find that constraint programming fails to solve the TAP mainly because of the stability constraints. Instead they provide a a mixed integer formulation for the problem and use a commercial solver to solve it. However, their running times are much too long to allow this method to be used in a tactical setting and they specifically advocate for development of a heuristic method for determining feasibility of the TAP. In this paper, we update and modify their method and this yields a significant improvement on running times. However, even with this improvement running times are still a bit too long. Neo et al. (2006) solves the integrated problem of routing a fleet of multi-compartment tankers with the tank allocation aspect included. They present an integer programming model for this problem and use a commercial solver to solve it. However, even for a small instance with just a single ship and only 10 potential cargoes, they report running times above 18,000 seconds. The aim of our work is therefore to develop a heuristic method that can facilitate the incorporation of the tank allocation aspect into the routing and scheduling planning phase by efficiently finding feasible cargo allocations for given ship routes.

In this paper we explore the TAP from a tactical perspective where the main objective is to quickly assess feasibility of a given ship route rather than finding an optimal tank allocation. Our main contribution is a heuristic solution method for efficiently finding feasible cargo allocations. Computational results show that it can solve 99% of the considered instances within 0.4 seconds and all of them if allowed longer running time. The heuristic does however struggle on two instances causing an overall longer average running time than found with an optimality based method from

the literature. However, when running time is below 10 seconds, our heuristic clearly outperforms the optimality based method by consistently solving more instances. Two further contributions of this work are therefore the modification of this optimality based method and a hybrid method that combines the developed heuristic with this modified method. Computational results show that on the considered instances this hybrid method cuts between 90% and 94% of average running times compared to the other methods and consistently solves more instances than the other methods within any given time limit. The average running time for the hybrid method is just 0.027 seconds. Hence, we have developed a solution method that is efficient enough to allow the inclusion of the tank allocation aspect into the routing and scheduling phase.

The remainder of the paper is organized as follows. Section 2 provides a problem description as well as a mathematical model for the problem and gives an overview of the existing literature related to the TAP. The devised heuristic is described in Section 3, while section 4 describes the data used to evaluate its performance. In Section 5 we tune the heuristic and in Section 6 we compare it to an optimality based method and a modified version of this method. Further, we explore the effect of combining our heuristic with the modified version of the optimality based method. Finally, concluding remarks and suggestions for future work are discussed in Section 7.

2 Problem description

In this paper we consider the TAP as it is described in Hvattum et al. (2009). However, we approach the problem on a tactical level where the focus is on feasibility rather than optimality.

For each instance, a fixed route is given for a ship with a number of tanks (or compartments). Besides the capacity of a tank, also the material and the coating can have an impact on how it can be used. A ship route is a collection of voyage legs and each leg has a departure port where one cargo is picked up and a destination port where one cargo is discharged, though not necessarily the same cargo as the one that was picked up. The destination port of one leg is the departure port of the next leg. Note that both the heuristic and hybrid method described in Section 3 also work in situations where multiple cargoes can be picked up and discharged in each port.

A set of cargoes that must be carried is also given. For each cargo, the volume and density is given, and also its pickup and discharge ports. If it is allowed to move cargoes between tanks after they have been loaded onboard the ship, the full route problem can be reduced to multiple instances of the single instance tank allocation problem where the allocation problem is solved for a set of cargoes on a single voyage leg. However, normally it is not allowed to move cargoes once they have been allocated and certainly, it will always be undesirable to so, due to both time consumption and possibly additional tank cleaning costs. Therefore, we assume that it is not allowed to move cargoes once they have been allocated. When we define a problem instance, some cargoes may already be onboard the ship at the beginning of the planning period and thereby occupy some of the tanks.

The tactical version of the TAP can now formally be defined: Given a ship, a ship route and a set of cargoes, find a feasible allocation of cargoes to tanks on the ship. A tank allocation is constrained by a number of factors that can be divided into three main groups:

- **Tank usage**: Each cargo must be allocated to one or several tanks and the cargo volume cannot exceed the total capacity of the allocated tank(s). A cargo can only go into a tank if it is compatible with the coating. It is not allowed to mix different products in the same tanks. Even if multiple cargoes consists of the same material, they cannot be mixed but must remain in separate tanks. For liquid products, a minimum volume must be allocated to used tanks in order to avoid excessive sloshing at sea.
- **Ship stability**: Ship stability and strength must be maintained throughout the route.
- **Hazmat rules**: Due to hazmat rules, certain materials cannot be allocated to neighboring tanks and certain materials cannot follow each other in a tank unless it is cleaned.

For completeness we here present the model derived and thoroughly described in Hvattum et al. (2009), though with slightly modified notation. We have the following sets:

C	Set of cargoes
T	Set of tanks
C_c^C	Set of cargoes in conflict with cargo c
C_t^T	Set of cargoes compatible with tank t
T_c^C	Set of tanks compatible with cargo c
T_{ckt}	Set of tanks which cannot be used for cargo k if cargo c is in tank t
N_c	Set of cargoes on board the ship immediately after adding cargo c
P_c	Set of all cargoes that have been present on the ship before adding cargo c
Q	Set of subsets; each set corresponds to the cargoes present on the ship at a time when the stability and strength restrictions apply.
S	Set of stability and strength dimensions, e.g. trim and roll

For each cargo c, the cargo volume v_c and the cargo density d_c is given. The capacity of tank t is denoted κ_t while the minimum volume in the tank when used is denoted b_t. Furthermore, m_t^s denotes the moment arm for tank t with respect to stability or strength dimension $s \in S$ while m^{s+} and m^{s-} denote, respectively, the upper and lower limit on total moment for stability or strength dimension $s \in S$. Finally, h_{ckt} denotes the minimum number of compatible cargoes that must be allocated to tank t before cargo c can be allocated to tank t if a cargo $k \in C_c^C$ that is incompatible with cargo c has previously been allocated to tank t.

There are three sets of decision variables. The first, x_{ct}, is a binary decision variable equal to 1 if cargo c is allocated to tank t. The second, y_{ct}, is a continuous variable indicating the volume of cargo c in tank t. Finally, z_{ct} is a binary decision variable, which is equal to 1 if tank t is cleaned just before adding cargo c. A feasible solution can now be defined by the following constraints:

$$y_{ct} \leq \kappa_t x_{ct}, \qquad \forall c \in C,\, t \in T_c^C, \qquad (1)$$

$$b_t x_{ct} \leq y_{ct}, \qquad \forall c \in C,\, t \in T_c^C, \qquad (2)$$

$$\sum_{t \in T_c^C} y_{ct} = v_c, \qquad \forall c \in C, \qquad (3)$$

$$\sum_{k \in C_t^T \cap N_c} x_{kt} \leq 1, \qquad \forall c \in C,\, t \in T, \qquad (4)$$

$$\sum_{k \in C_c^C \cap N_c} \sum_{u \in T_{ckt}} x_{ku} \leq M_{ct}(1 - x_{ct}), \qquad \forall c \in C,\, t \in T_c^C, \qquad (5)$$

$$m^{s-} \leq \sum_{c \in R} \sum_{t \in T_c^C} m_t^s d_c y_{ct} \leq m^{s+}, \qquad \forall R \in Q,\, s \in S, \qquad (6)$$

$$h_{ckt}(x_{ct} - z_{ct}) - \sum_{j \in R} (x_{jt} + h_{ckt} z_{jt}) \leq h_{ckt}(1 - x_{kt}), \qquad \forall c \in C, t \in T_c^C, k \in P_c \cap C_c^C, R = P_c \setminus P_k \setminus \{k\}, \qquad (7)$$

$$x_{ct} \in \{0, 1\}, \qquad \forall c \in C,\, t \in T_c^C, \qquad (8)$$

$$y_{ct} \geq 0, \qquad \forall c \in C,\, t \in T_c^C, \qquad (9)$$

$$z_{ct} \in \{0, 1\}, \qquad \forall c \in C,\, t \in T_c^C. \qquad (10)$$

Due to our focus on feasibility, we have not presented an objective function. However, a possible objective function could be the minimisation of tank cleaning or the maximisation of vacant tank capacity in order to preserve flexibility to accommodate future cargoes or cargo changes.

Constraints (1) ensure that the capacity of each tank is not exceeded, and that only cargoes allocated to a given tank can be put into that tank. Constraints (2) ensure that an occupied tank is allotted a minimum content to avoid sloshing. In addition, constraints (3) and (4) make sure that the entire cargo is placed on the ship and prohibit more than one cargo per tank at a time. Constraints (5) make sure that the hazmat rules regarding what chemicals can be put next to each

other is maintained. Here, M_{ct} is a large constant which can be set as $\max_k\{T_{ckt}\}, c \in C, t \in T_c^C$. Constraints (6) ensure that the ship remains sufficiently balanced with regards to roll, trim and strength on all legs of the route. Note that these constraints are simplified versions of the actual non-linear expressions. Hvattum et al. (2009) use this same simplification and argue that the loss from it is small. Constraints (7) restrict a cargo c from going into a tank t if the previous cargo k is incompatible with c. Exceptions to this can be allowed if the tank is cleaned or h_{lkt} other cargoes occupy the tank in between k and c. Note that by ignoring the tank cleaning costs, we can actually ignore these constraints and simply assume that all tanks are cleaned. However, we have included them here for the sake of completeness. Finally, constraints (8), (9), and (10) formally define the variables of the problem. In (Hvattum et al., 2009) a complexity analysis concludes that the TAP is *NP-Complete* by showing that it is a generalisation of the segregated storage problem that has been shown to be *NP-Complete*.

A more simple variant of the problem was already introduced in (Vouros et al., 1996). Here an expert system is presented for the task of allocating cargoes to a ship for one port. So, given the current configuration how does one load a cargo on board the ship in the best possible way adhering to ship stability constraints and hazmat rules. The paper only describes the approach, neither testing nor implementation details are presented.

As we have already mentioned a number of times, Hvattum et al. (2009) first formally describe the variant of the TAP that we base our work upon. They solve their mixed integer formulation with a commercial solver but report running times that are far beyond what would be acceptable in a tactical setting. They also try to solve the problem using constraint programming. However, their constraint solver failed to find a feasible solution to even one of the instances. Therefore, they note in their conclusion that a heuristic method for this problem is of interest. The first description of the routing problem for multi-compartment tankers is given in Jetlund and Karimi (2004). This paper presents the problem of optimising the route of a single multi-compartment tanker as well as a fleet of ships. Although the paper presents the routing problem for a multi-compartment tanker, the tank allocation component is omitted according to the authors because it is not important when looking at the problem from a strategic setting. An integer programming model for both a single ship and for a whole fleet is proposed and solved using a commercial solver. Important problem characteristics such as cargo compatibility and ship stability are not mentioned in this paper but are part of Neo et al. (2006). Here an integer programming model is also the main contribution. Now the model includes the aforementioned constraints on cargo compatibility and ship stability. It is solved using a commercial solver and exhibits very large running times. For a single ship, 10 compartments and 10 potential cargoes, running times are above 18,000 seconds. In the same line of research, Coccola and Mendez (2013) consider basically the same integer programming model as in the previous papers. This approach does not take ship stability and hazmat constraints into consideration. The paper contains the description of an iterative heuristic aimed at situations where excessive running times prohibit optimisation for an entire fleet. The problem is decomposed into single ship problems and cargo conflicts are resolved solving a 2-ship problem in an iterative manner. Authors report a 40% improvement in profit over the manual plan on a single instance and a "significant" improvement over "other" approaches.

Kobayashi and Kubo (2010) consider a tanker problem, though without hazmat and stability constraints. They decompose the problem into a tank allocation problem and a routing problem. Both problems are modeled and solved as set partitioning problems, but for the routing problem the number of columns necessitates column generation. Using column generation to solve the LP relaxation, an integer solution is afterwards found by branch-and-bound. In addition, Wu et al. (2011) describe a decision support system (DSS) that includes the relevant real life constraints and a graphical user interface to deliver a real-life applicable DSS. The optimization is based on a relatively simple heuristic but includes routing and allocation constraints in an integrated fashion. The inclusion of hazmat constraints is, however, not obvious and the method only works by inserting new cargoes to an already existing schedule. Oh and Karimi (2008) describe another DSS. They assume that ship stability can always be achieved by using ballast tanks. They optimise for a fleet of multi-compartment tankers by decomposing the problem into a tank allocation problem and a routing problem. A simple enumerative approach is used for generating the routes for the set partitioning problem. Cargo allocation is done using a heuristic which is not described in any detail.

The above papers allow cargoes to be picked up or rejected, which implies profit maximisation instead of a feasibility focus. In contrast, Schaus et al. (2012) assume a fixed route, just as (Hvattum et al., 2009), and also focus on feasibility. They consider a set of cargoes to be allocated without any notion of route and therefore loading and unloading along the route. The paper does not look at ship stability but does incorporate hazmat constraints, compatibility with previous cargoes, and the sealing on the tanker walls in a constraint programming approach.

In (Fagerholt and Christiansen, 2000) a tramp ship problem with adjustable compartments is discussed. Here the cargo hold can be partitioned into smaller holds by inserting bulkheads at a discrete number of positions. The problem is a combined routing and allocation problem for a fleet of ships but ship stability and hazmat constraints are not considered. The solution approach is based on an iterative heuristic using a priori generation and dynamic programming to determine the optimal schedules. Each iteration uses a variant of the Traveling Salesman Problem (incorporating allocation, time windows and precedence constraints) to construct ship routes.

Another important maritime stowage problem is the container stowage problem found in liner shipping. The two problems share some of the same components such as ship stability and spatial separation requirements for dangerous goods. Also, finding a feasible stowage plan is often more important than finding an optimal one. However, there are important operational differences that call for tailor made solution methods for each of these shipping segments. To mention a few, we note that container stacking calls for efficient cargo handling to minimize container shifting when loading and unloading and this is of course not an aspect in bulk shipping. On the other hand, in the container stowage problem there is no need to consider individual tank constraints, such as e.g. tank capacity and tank coating. For further information on the container stowage problem and solution approaches see e.g. (Martin et al., 1988; Wilson and Roach, 2000).

Transportation in compartments also exists in road transport. Most applications come from distribution of petroleum products; other examples are waste collection and food. In road transport, compartments can be fixed in size or flexible. In (Derigs et al., 2011) a survey and computational comparison of current literature within road transport is presented. Interestingly, road transport always assumes integrated optimization of routing and cargo allocation, and, in addition, the majority of developed approaches are based on heuristics or metaheuristics. Quite often when multiple compartments are considered in the literature, the objective is to just deliver from a central depot to e.g. petrol stations (Cornillier et al., 2009) and not to combine both pickup and delivery as we do here. Furthermore, in these problems bulk ship constraints such as stability are not considered.

The *segregated storage problem* is another related compartment oriented storage optimization problem. In this problem a silo with a number of compartments has to be filled with grain and only one type of grain can go into each compartment. However, contrary to our problem, identical "cargoes" can be mixed into the same compartment. In (Barbucha and Filipowicz, 1997) the segregated storage problem and some variants of the basic problem are compared to storage problems in transport in general but also in particular to maritime transportation. The complexity of the different variants is described and a few numerical examples are given.

3 Solution Method

The aim of our work is to develop an efficient method for finding feasible cargo allocations in a very short amount of time. Finding a good allocation is left for operational planning where the solution process proposed by Hvattum et al. (2009) is sufficient with respect to time. Our search for a feasible allocation assumes that initial crude feasibility tests have been performed, e.g. verifying that total cargo volume during each voyage leg does not exceed ship capacity.

When we focus on feasibility, we can disregard tank cleaning costs. This means that any two cargoes can potentially follow each other in a tank as we can just pay the cost of cleaning. Thereby, we no longer have to keep track of previous tank allocations. Furthermore, if the ship is at some point empty during its route, we can split the problem into smaller *subproblems* or *subinstances* since any voyage leg prior to the ballast leg is independent of any voyage leg after the ballast leg.

Even though we disregard cleaning costs, the decisions at any voyage leg still affect decisions at future voyage legs since we are not allowed to move a cargo once it has been allocated. Consequently, allocating cargoes one by one as they are picked up along the route will be a bad idea. We

can easily end up allocating a cargo to tanks that are required for a feasible allocation of another cargo picked up later. Our heuristic is therefore based on a priority ordering of cargoes that defines the order in which they will be allocated one by one. The function *selectCargo()* returns the next cargo to allocate. Each time a cargo has been allocated, the chosen tanks are made unavailable for all cargoes onboard with the considered cargo as well as neighboring tanks if there are any conflicting cargoes. The priority ordering for all cargoes affected by the chosen allocation is then updated before selecting the next cargo to allocate.

Ship stability at a voyage leg cannot be calculated before all cargoes onboard during the leg have been allocated. Therefore, when allocating cargoes one by one, we initially only use an estimate of ship stability based on allocated capacity rather than on the actual cargo amount allocated to each tank. This means that we initially only reserve sufficient tank capacity for one cargo at a time using a function called *selectTanks()* which takes a specific cargo as its parameter. If we manage to find sufficient capacity for all cargoes, we check if there exists a combination of cargo amounts to tanks, which can secure ship stability during the entire route. This is done by solving a simple linear program (LP) where each cargo is predefined to be allocated to specific tanks, i.e. solving (1), (2), (3), (6) and (9) with all x variables fixed. The function *solveLP()* performs this task. Note that ballast tanks can easily be incorporated by including them in these LPs.

3.1 Diversification

The quality of the heuristic will naturally depend on the quality of the two procedures for selecting the next cargo to allocate and for selecting tanks when reserving capacity for the selected cargo. We have chosen to keep these two procedures simple and instead introduce randomness in each of them to produce diverse results with each call to the heuristic. In Section 3.2 we describe how randomness is introduced into the cargo selection procedure while in Section 3.3 we describe how it is introduced into the tank selection procedure. Within a given time limit, we then allow the heuristic to restart each time it fails to find a feasible allocation for a given subinstance, whether the infeasibility comes from some cargoes not being fully allocated or from instability when determining cargo amounts. Restarting simply means that all cargoes that have already been allocated, are removed from the ship again. The basic outline of the heuristic can be seen in Algorithm 1.

3.2 Cargo priority - function *selectCargo()*

We prioritise cargoes based on the ratio of the volume of each cargo and the amount of available tank capacity for this cargo. A tank is available if it is both compatible and vacant, also with respect to neighbor cargoes. Obviously, the closer to one this number gets, the more important it is to allocate the cargo, since only few of the available tanks can be occupied by other cargoes before it becomes impossible to allocate the given cargo. For a cargo i we denote this estimate of importance by I_i. The basic idea of the cargo selection procedure is then to iteratively select the cargo with the highest value of I_i. Note that if $I_i > 1$, it means that due to already allocated cargoes, there is not enough available tank capacity left to allocate cargo i and so, we must restart.

In order to diversify the results of the heuristic, we add some randomness to this otherwise deterministic procedure. We do this by allowing the procedure to sometimes discard the most important cargo and instead select a less important cargo. However, if the most important cargo, cargo i, has I_i above a threshold value that we denote I, we always select this cargo. The reason for this threshold approach is that a cargo i with high I_i will only be able to spare few of its available tanks to other cargoes before it becomes impossible to allocate cargo i. Therefore, it will most often be best to allocate cargo i first. The most important cargo, cargo i, with $I_i < I$ will now be selected with a probability of P. We tune the parameters I and P in Section 5. Preliminary tests show that the deterministic cargo selection performs best and so, we want to ensure that this approach is followed before adding randomness. Therefore, the first run of the heuristic will use the deterministic selection procedure while the randomised approach will be used after restarting. The basic outline of the cargo selection procedure is then as presented in Algorithm 2 where $\mathcal{C}[0]$ denotes the first element in the list $\mathcal{C}$ and $rand(0, 1)$ denotes a random number between 0 and 1.

Algorithm 1: Basic heuristic outline

```
determine initial cargo priority ordering;
split problem to create list, Subs, of smaller subinstances;
while Subs ≠ ∅ AND time limit not exceeded do
    Sub ← first subinstance in list Subs;
    while no stable allocation found for Sub AND time limit not exceeded do
        while cargoes in Sub remain to be allocated AND time limit not exceeded do
            cargo i ← selectCargo();
            if not enough available tank capacity to allocate cargo i then
                if time limit not exceeded then
                    restart Sub;
                else
                    stop;
            else
                selectTanks( cargo i );
                update available tank capacity for affected cargoes;
                update priority ordering of unallocated cargoes;
        if all cargoes in Sub fully allocated then
            solveLP();
            if allocation is unstable then
                if time limit not exceeded then
                    restart Sub;
                else
                    stop;
            else
                remove Sub from Subs;
```

Algorithm 2: Outline of cargo selection procedure, i.e. function *selectCargo()*

```
C ← list of all unallocated cargoes sorted in order of decreasing importance;
cargo i ← C[0];
if I_i > I or we did not restart then
    return cargo i;
else
    while C contains at least 2 cargoes and rand(0,1) > P do
        erase C[0] from C;
    return C[0];
```

3.3 Reserving sufficient capacity - function *selectTanks()*

We allocate each cargo as stable as possible by iteratively choosing tanks that have moment arms in the opposite direction of the ships current stability and strength estimates. Assume for example that the current condition of the ship is that its stability and strength measures are positive for roll, negative for trim and negative for strength. Then we only consider available tanks with negative moment arm for roll and positive moment arm for both trim and strength. Naturally, if no such tanks remain, we look for tanks that fulfill two out of these three moment characteristics and we continue this process of lowering our requirements until we find available tanks. During preprocessing, we therefore group the tanks according to their moment arms for the various stability and strength dimensions. E.g. considering roll, trim and strength, we obtain 27

such *stability dimension groups* (SDG) since a moment arm can also be neutral, i.e. equal to 0. To determine the correct SDG to consider, we have a function called $selectSDG()$ that, given the current condition of the ship, tells us which SDG to consider. Note that multiple SDGs can be eligible if one of the stability and strength measures is currently neutral or if a preferred SDG does not contain any available tanks. In such cases we break ties arbitrarily. Once a tank is selected, we can determine if further capacity is required to fully allocate the cargo, and if there is, we update the stability and strength measures to determine a new SDG of eligible tanks to choose from.

In principle each cargo does not by itself have to be allocated in a stable manner since cargoes can outbalance each other. However, many cargoes are at some point onboard the ship alone ("alone-cargoes") and must therefore be allocated in a stable manner. Thereby, any cargo that is at some point onboard the ship with only an alone-cargo, must also be allocated in a somewhat stable manner. This reasoning can continue, thus motivating the search for stable allocations for each individual cargo. Furthermore, allocating each cargo in a stable manner yields robust plans since the stability of the ship does not rely on cargoes to outbalance each other. Hence, any cargo can in principle be removed from the route without causing instability. Note however, that we do not *require* each cargo to be allocated in a stable manner on its own but simply *seek* to find a stable allocation for each cargo. Often it is not possible to find stable allocations for the individual cargoes and furthermore, the added randomness can easily cause us to reject them if they exist.

Finding a feasible allocation for all cargoes relies on a clever allocation of each individual cargo as the cargoes are in effect competing for the same tanks. For cargoes not in conflict with other cargoes, a clever allocation mainly relates to capacity utilisation of the chosen tanks. For cargoes in conflict with other cargoes, it is however equally important to confine the cargo to smaller groups of tanks as opposed to scattered all over the ship whereby a lot of neighboring tank capacity becomes unavailable for cargoes in conflict with this cargo. So, we distinguish between two cargo types:

> A cargo i is a *conflict cargo* if there is at least one other cargo j that is in conflict with cargo i and where cargo j has not yet been allocated.
>
> A cargo i is a *non-conflict cargo* if it is not in conflict with any other cargoes or if all conflicting cargoes have already been allocated.

When allocating a non-conflict cargo, we then focus on capacity utilisation and simply sort the compatible and vacant tanks within each SDG by size in decreasing order. When allocating a conflict cargo we instead sort the compatible and vacant tanks within each SDG by decreasing ID, which reflects their location on the ship. This way we will choose tanks grouped together although we might use several groups of tanks in different locations of the ship to secure stability and strength of the ship. In Figure 1 we give a small example of how the ID's reflect the location of the tanks onboard the ship. Note that with this very crude estimate of neighboring tanks and groupings, our heuristic will correctly assume that tanks 1 and 3 are neighbors to tank 2 but it will have no idea that tank 5 is also a neighbor to tank 2. Furthermore, it will assume that tanks 3 and 4 are neighbors, which clearly they are not. This sort of ship specific information could obviously be used to improve the tank sorting function. However, since the tanks are sorted within each specific SDG, the tank sorting function does not reflect neighboring tanks across different SDGs. Thereby, the effect of improving the tank sorting function might be limited and furthermore it would sacrifice simplicity and make the heuristic much less generic than when we do not use the actual layout of the considered ship. Therefore, we have chosen to use this crude estimate.

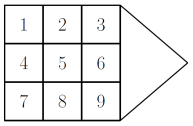

Figure 1: Example of tank IDs to reflect ship layout

For all cargoes we use a capacity utilisation threshold so that no tank can be selected if the resulting allocation has a capacity utilisation less than this threshold. However, we allow this threshold to vary with the cargo since cargoes onboard during crowded voyage legs must utilise

capacity better than cargoes only onboard during less crowded legs. We therefore determine the minimal capacity utilisation of each voyage leg by dividing the total onboard cargo volume by ship capacity. For each cargo, we then consider all the voyage legs that the cargo is onboard the ship during, and the maximal of the minimal capacity utilisations corresponding to these specific voyage legs, is then the capacity utilisation threshold for this particular cargo. We denote the capacity utilisation threshold for cargo i by U_i. If we want to generate solutions with high capacity utilisation, we can simply exchange this cargo specific threshold by a sufficiently high uniform threshold. This approach will utilise ship capacity much better than when using cargo specific thresholds and thereby also better preserve flexibility to accommodate future cargoes. However, it will increase the running time of the heuristic and since we only care about feasibility, we use the cargo specific thresholds. Naturally, we might have to lower this threshold gradually if no tanks remain that fulfill the utilisation requirement. In cases where multiple SDGs are applicable, we only lower the utilisation threshold if none of the SDGs contain an eligible tank. For conflict cargoes, choosing groups of tanks is important and as soon as we start discarding tanks in the otherwise location based ordered tank list, we are in effect choosing tanks that are scattered. Therefore, gradually decreasing the capacity utilisation requirement with only small decrements does not seem a good idea for conflict cargoes. As soon as we discard a tank due to capacity utilisation, to avoid discarding many tanks, we should instead lower this requirement almost to a point that it becomes obsolete. For non-conflict cargoes this, however, is not necessary. Therefore, we use separate parameters for the two cargo groups for the incremental lowering of the capacity utilisation threshold when we run out of tanks. For non-conflict cargoes and conflict cargoes we respectively denote these ΔU_{NC} and ΔU_C and tune them in Section 5.

To ensure diversification we add elements of randomness to the otherwise deterministic tank selection procedure. Similar to the cargo selection procedure, we do this by only accepting a given tank with a certain probability. To keep the parameter list as short as possible, we reuse the P from the cargo selection procedure. For the conflict cargoes, we again need to ensure that we choose tanks in small groups rather than tanks scattered all over the ship. Therefore, for conflict cargoes we use an additional parameter, ΔP, that is used to iteratively increment the tank acceptance probability each time a tank has been discarded. Thereby, when a tank is discarded, we lower the probability of discarding the next tank in the list. All discarded tanks are stored in case we run out of available tanks and must return to these. The discarded tanks are also grouped according to their stability dimensions and sorted according to either size or ID, depending on the cargo type.

Preliminary tests show that for conflict cargoes it is best to use the deterministic tank selection procedure prior to restarting and the randomised approach afterwards and so, we use this approach. This confirms that the deterministic approach best enforces the group wise selection criteria. However, for non-conflict cargoes preliminary tests do not indicate any benefits from using the deterministic procedure and so, for these cargoes, we always use the randomised approach.

Each restart requires all allocated cargoes to be reallocated and this is obviously much more time consuming than just restarting for a single cargo, i.e. removing this cargo from the ship and reallocating it. Therefore, iteratively allocating the cargoes and then restarting the whole process in case of infeasibility is not an attractive approach. Instead, each time we allocate a cargo, we iteratively reallocate it until the quality of the allocation is sufficient. Keeping the strict computation times in mind, we use a very crude estimate of quality by simply requiring the capacity utilisation of an allocation for cargo i to be at least as high as the capacity utilisation of the most crowded voyage leg, that cargo i is onboard during, i.e. at least as high as U_i. We use the parameter L to limit the number of allowed reallocations of each cargo. If we reallocate as many times as allowed without finding a good enough allocation, we use the best one found so far.

Since the tank selection procedure for conflict cargoes is deterministic before restarting, there is no point in reallocating conflict cargoes before restarting. After restarting, conflict cargoes are reallocated, just as non-conflict cargoes always are. However, preliminary tests also show that after restarting it can be beneficial to allow the first iteration of the reallocation procedure for conflict cargoes to follow the deterministic tank selection procedure. Reusing the P parameter, we then only perform the first iteration of the reallocation procedure for conflict cargoes, i.e. the deterministic one, if a randomly generated number between 0 and 1, $rand(0, 1)$, is less than P.

Algorithm 3 shows the outline of the tank selection procedure for a generic cargo i. Here, ΔU_i is either equal to ΔU_C or ΔU_{NC} depending on whether cargo i is a conflict cargo or not.

Algorithm 3: Algorithm for selecting tanks for cargo i, i.e. function *selectTanks(cargo i)*

```
if cargo i is non-conflict OR we have restarted then
    if cargo i is non-conflict OR rand(0,1) < P then
        count ← 0;
    else
        count ← 1 (i.e. skip deterministic iteration);
else
    count ← L (i.e. don't reallocate);
while count ≤ L do
    create size/ID sorted list of compatible and vacant tanks for each SDG;
    u ← U_i;
    stability estimate e ← 0;
    clear allocation and discarded tank list from last iteration;
    p ← P;
    while further capacity required do
        G ← selectSDG(e);
        tank t ← first tank in sorted tank list of G;
        while tank t ≠ NULL, i.e. tanks remain do
            if capacity utilisation ≥ u then
                remove tank t from list of tanks from G;
                if not using discarded tanks AND (cargo i is non-conflict OR (using
                reallocation AND count > 0)) then
                    num ← rand(0,1);
                else
                    num ← 0 (i.e. no tanks are discarded);
                if num < p then
                    allocate cargo i to tank t;
                    update e using capacity of tank t;
                    tank t ← NULL, i.e. end loop to allow switching to new SDG;
                else
                    add tank t to size/ID sorted list of discarded tanks for G;
                    tank t ← next tank in sorted tank list of G;
                    if cargo i is conflict cargo then
                        p ← p + ΔP;
            else
                tank t ← next tank in sorted tank list of G;
        if no tank was found AND no other SDGs are applicable then
            if tanks remain that are not discarded then
                u ← u − ΔU_i;
            else
                from discarded tanks, create size/ID sorted tank list for each SDG;
                u ← U_i;
    if using reallocation then
        if capacity utilisation ≥ U_i then
            count ← L, i.e. exit reallocation loop;
        else
            update best allocation found so far;
    count++;
if using reallocation AND reallocated as many times as allowed then
    use best allocation found;
```

4 Data

In (Hvattum et al., 2009), an instance generator is developed to create a varied set of realistic and feasible TAP instances based on two real tank ships. The instance generator is used to generate 720 data instances and we base our computational study on these instances. Here, we briefly describe the main features of the data instances, while a thorough description of the instance generator can be found in (Hvattum et al., 2009).

There are two tank ships of different size and configuration. The smaller ship has 24 tanks and can carry up to 10,000m^3 while the larger ship has 38 tanks and a capacity of 45,000m^3. There are two types of tanks, namely stainless steel tanks and tanks with a zinc silicate coating. The data instances only contain data for stability and strength restrictions with respect to roll. This means that, although our heuristic is generic enough to handle several stability and strength dimensions, we will here only consider the roll dimension and therefore only have three stability and strength dimension groups, containing, respectively, tanks with negative, neutral and positive moment arm for roll. There are three categories for cargoes:

1. cargoes that can go into any tank and do not conflict with any other cargoes
2. cargoes that can go into any tank but which conflict with all category 3 cargoes
3. cargoes that can only go into tanks with zinc silicate coating and which conflict with all other cargoes in category 2 and 3

Cargoes that are in conflict with each other cannot simultaneously occupy tanks that share a side, i.e tanks that are neighbors.

In each data instance the first ten generated cargoes are defined to be locked, i.e. they are already allocated and cannot be moved. They simply act as the history of the ship and cause some tanks to be occupied when planning starts. Hvattum et al. (2009) generate instances with, respectively, 20, 30 and 40 cargoes where the first ten are locked cargoes. To create a varied set of instances, they use three additional parameters:

D denoting the probability distributions for the category of each load. They allow four settings for this parameter and label them $D1(0.6, 0.4, 0.0)$, $D2(0.5, 0.4, 0.1)$, $D3(0.4, 0.4, 0.2)$ and $D4(0.3, 0.4, 0.3)$. Here $D1$ refers to the case where 60% of the cargoes are expected to come from category 1, 40% from category 2 while no cargoes are expected from category 3.

F denoting the minimum/maximum capacity utilisation of the ship before loading/unloading. This parameter is allowed three settings which are labeled $F1(0.65, 0.35)$, $F2(0.75, 0.25)$ and $F3(0.85, 0.15)$. Here $F1$ refers to the case where the ship will visit pickup locations until the the total load exceeds 65% of ship capacity and then start to visit discharge locations until the total load becomes less than 35% whereafter the ship will again visit pickup locations.

V denoting the distribution of load volumes. This parameter is allowed three settings labeled $V3(1000 - 5000m^3)$, $V6(3000 - 9000m^3)$ and $V12(8000 - 16,000m^3)$. Here $V1$ denotes the case where cargo sizes follow a uniform distribution over the interval 1000 to 5000m^3.

Combining ship type, number of cargoes and the above three parameters while ruling out unrealistic combinations, Hvattum et al. (2009) obtain 144 parameter setting groups. For each of these 144 groups, they randomly generate five feasible instances yielding a total of 720 instances for which the existence of feasible solutions is verified. We let TAP_T24C20D3F1V3_01 denote the first generated instance for the 24 tanks ship with 20 cargoes (including ten locked cargoes) and with parameters D, F and V set to, respectively, 3, 1 and 3.

4.1 Instability for locked cargoes

Since solving any one of the above 720 TAP instances only entails finding a feasible allocation for the unlocked cargoes, the feature in the instance generator that verifies the existence of a feasible solution, does not check the allocation of the locked part of the instance. This unfortunately means that for quite a few of the 720 instances, the ship is at some point unstable during the locked part

of the ship route. Solving the problem with a mixed integer formulation and CPLEX, as done in (Hvattum et al., 2009), this instability is not a problem since the entire solution space is explored, including solutions where one cargo can be allocated solely to one end of the ship in order to outbalance a locked cargo placed in the opposite end of the ship. However, since our heuristic tries to allocate each individual cargo in a stable manner, it will not perform well on instances with built-in instabilities, i.e. instances where the locked part contains cargoes or groups of cargoes that are placed in an unstable manner. Note that even though the heuristic *seeks* stable allocations for each cargo, it *can* allocate cargoes in an unstable manner and thereby explore the entire solution space. However, the chances of the heuristic finding extreme solutions are slim and thereby we are penalised for an instability created before the ship was even handed over to us. If our heuristic were used continuously, then all cargoes previously allocated, i.e. locked cargoes, would also be allocated in a somewhat stable manner and then this would not be a problem. Therefore, a computational study using these instances can, in this respect, be expected to yield a conservative estimate of the success rate of our heuristic. In order to get a feel for the effect of excluding such built-in instabilities, we consider three distinct sets of data instances during our computational study:

Set_{720} Contains all 720 instances

Set_{648} Contains the 648 instances that are stable when the ship is "handed over" to us, i.e. just before picking up cargo number 11

Set_{486} Contains the 486 instances that are stable during the entire locked part of the last subinstance before the ship is "handed over" to us, i.e. during the first subinstance that we have to solve. Note that we do not require the ship to be stable during the entire locked part of the route as we do not care about what happens before the first subinstance that we have to solve. I.e. if the ship is unstable and then becomes empty before allocating other locked cargoes that we have to take into account, then we do not care about the instability

To properly understand the reason for including Set_{486}, consider the example illustrated in Table 1. Here, 'Action' refers to the port action taken at a particular point in time and time progresses as we move to the right in Table 1. 'Onboard' refers to the cargoes onboard immediately after this action while 'Roll' refers to the roll measure of the ship immediately after the action. We let $-R$ and R denote, respectively, the lower and upper threshold for stability in the roll dimension. For port actions, '9+' and '9-' refer to the actions of respectively loading and discharging cargo number 9, and similarly for higher numbers. We assume that the ship becomes empty just before loading cargo 9 whereafter cargo 9, 10, 11, and 12 are all picked up and then again discharged in the same order as they were picked up. This means that the first subinstance that we have to solve, contains the locked cargoes, cargoes 9 and 10, and the two unlocked cargoes, cargoes 11 and 12, whereby we can confine our example to include just the small part of the ship route given in Table 1. Now assume that cargo 9 is allocated in an unstable manner and that cargo 10 is allocated in a similarly unstable manner but that these two cargoes outbalance each other so that the ship is stable when it is handed over to us, i.e. just before picking up cargo 11. So, assume that the allocation of cargo 9 creates a roll measure of $-R-1$ and that the allocation of cargo 10 creates a roll measure of $R+1$. The roll measure will then be 0 when the ship is handed over to us. Then our heuristic will try to allocate cargoes 11 and 12 in a stable manner and we can, for argument's sake, assume that the effect of their allocations on the roll measure is zero. After the pickup of cargoes 11 and 12, we will discharge cargo 9 and thereby obtain a roll measure of $R+1 > R$. This means that even though the ship was stable when handed over to us, and we

Table 1: Small example to illustrate reason for considering Set_{486}

Action	9+	10+	11+	12+	9-	10-	11-	12-
Onboard	9	9,10	9,10,11	9,10,11,12	10,11,12	11,12	12	-
Roll	$-R-1$	0	0	0	$R+1$	0	0	0

managed to allocate both cargo 11 and 12 in a fully stable manner, we ended up with an unstable ship. Theoretically, the heuristic could find a feasible solution to this problem. However, since the heuristic seeks to allocate each individual cargo in as stable a manner as possible, chances are that it will not find a feasible solution. Thereby, we are penalised for an instability created before the ship was even handed over to us. Assuming that the heuristic was used continuously, such instabilities would never occur and generally, we do not anticipate such instabilities in real life. Therefore, we find it reasonable to include Set_{486} in our computational study. Note however that even when confining our computational study to the 486 instances, there can still be instances that will affect the success rate of our heuristic in a negative manner compared to if our heuristic had been used continuously. Assume for example that we extend the subinstance in the example from above to include the locked cargo 8 and that cargo 8 and 9 are now both allocated in a manner that affects the roll measure with $-\frac{1}{2}R$, i.e. after picking up cargo 9, the ship will have a roll measure of $-R$ which just leaves the ship stable. Adding cargo 10 we obtain a roll measure of 1 which is again stable. Adding cargoes 11 and 12 maintains the roll measure at 1. However, once we discharge cargoes 8 and 9, we obtain a roll measure of $R+1$ and thereby an unstable ship even though the locked part contained no instabilities and we managed to allocate the unlocked cargoes in a fully stable manner. However, since such instances are per definition feasible and we do not want to exclude too many instances, we limit our computational study to the three sets above and simply note that the success rate of our heuristic on these sets can be expected to be a conservative estimate of the success rate that could be achieved if the heuristic was used continuously.

5 Tuning

Our heuristic has the following parameters, that must be tuned:

- P: probability of accepting a chosen cargo or tank or using the deterministic tank allocation for conflict cargoes
- I: threshold value for cargo importance that eliminates randomness in the cargo selection procedure
- ΔU_{NC} and ΔU_C: reduction parameter for non-conflict and conflict cargoes respectively. Used to iteratively lower the capacity utilisation threshold
- ΔP: increment parameter used to iteratively increase the probability of accepting a chosen tank for conflict cargoes
- L: number of times the heuristic is allowed to randomly reallocate a cargo

As discussed in Section 4, results from running our heuristic on the reduced data set Set_{486} yields the most realistic estimate of its performance compared to data sets Set_{720} and Set_{648} containing built-in instabilities. Therefore, during tuning we only consider the instances in Set_{486}. For the larger ship with 38 tanks, this set only contains 134 instances and these are again spread over a wide variety of instance types with varying cargo count, cargo sizes etc. As mentioned in Section 4, only five instances of each type are generated and with the removal of instances with built-in instabilities, this number is significantly lower for most instance types. Thereby, it will be difficult to extract only a subset of instances for tuning while at the same time retaining a diverse and representative sample of instances. Therefore, we have chosen to tune on all instances in Set_{486}; however, this will bias our conclusions slightly when performing the computational study on this data set in Section 6.

We tune two versions of the heuristic: One for the smaller ship with 24 tanks (denoted T24) and one for the larger ship with 38 tanks (denoted T38). During development of the heuristic and preliminary testing, we obtained qualified estimates for the initial parameter test value ranges. Within the initial interval for each parameter we chose three test values. With six parameters, each allowed three different values, we obtained $3^6 = 729$ parameter scenarios. Due to the randomness of the heuristic, we ran each scenario five times on each data instance in Set_{486} and evaluated the average performance of each parameter scenario. This resulted in a total of $5 \cdot 729 = 3645$ scenarios to be run on 486 instances. Since the heuristic is developed to efficiently solve numerous

subproblems in a tactical setting, we allowed a maximum run time of 0.2 seconds. On average, each scenario runs through all data instances in 3.3 seconds for T24 and 1.5 seconds for T38. Therefore, choosing the parameter setting with the best average performance from each of the six initial lists of test values for each ship took a total of $(3.3 + 1.5) \cdot 3645 = 17,496$ seconds, i.e. under 5 hours. The best choice of each parameter value allowed us to further reduce the initial test value ranges and repeat the process iteratively until the optimal parameter values were determined after three iterations. I.e. assuming that the initial test values for a parameter were $\{0.4, 0.6, 0.8\}$ and we concluded that 0.6 was the best of these, then in the next iteration we narrowed the range and instead tested $\{0.5, 0.6, 0.7\}$. A few times the parameter setting with best average performance was on the boundary of the test value range and so we re-expanded before again zooming in on the optimal parameter setting.

We encountered ties between multiple parameter scenarios several times during the tuning process. Since the average running time did not vary much between the different parameter scenarios, we broke ties by selecting the scenario with the most stable performance using the standard deviation of the solve count for the five sample runs as a measure of stability. The initial test values as well as the final chosen parameter settings are shown in Table 2 for both T24 and T38.

Table 2: Tuning results

	Initial test values	T24 Optimal value	T38 Optimal value
P	{0.4, 0.6, 0.8}	0.50	0.60
I	{0.4, 0.6, 0.8}	0.70	0.60
ΔU_{NC}	{0.05, 0.25, 0.45}	0.10	0.60
ΔU_C	{0.3, 0.5, 0.7}	0.45	0.45
ΔP	{0.05, 0.25, 0.45}	0.05	0.15
L	{20, 40, 60}	50	35

From the optimal parameter settings in Table 2, we note that P is between 50% and 60% meaning that the heuristic relies on a great deal of randomisation. Finally, we note that using the worst setting from the above test ranges resulted in a success rate that was only 1% lower than when using the optimal parameter setting. Therefore, the heuristic is quite robust to parameter changes.

6 Computational Results

In this section we evaluate the performance of our heuristic. We compare our results to the ones obtained in (Hvattum et al., 2009) from using an optimal method and to results from both updating and modifying their method. We also combine our heuristic with the both updated and modified version of the optimal method and evaluate the performance of this hybrid method.

6.1 Results from optimal method by Hvattum et al. (2009)

First, we briefly summarise the findings Hvattum et al. (2009) obtained solving their mixed integer program directly with CPLEX v.11 on an Intel 2.66GHz processor. In Table 3 we summarise their results for three different objective functions. The first objective function is to simply minimize 0, i.e. focus on feasibility, the second seeks to minimise tank cleaning while the last one maximises average capacity of vacant tanks during the ship's route. They allowed a maximum run time of 600 seconds to solve each instance in Set_{720}. To capture the variance in running times, Table 3 shows the number of instances for which a feasible solution is found within a given time limit stated in the top row as well as the average running time given in the last column.

On average, the method that focuses on feasibility, i.e. objective (1), is fastest. However, allowing running times of 10 seconds or more, the two other objective functions result in a greater number of feasible solutions. For all three methods, the average running time is too long for a

Table 3: Results from Hvattum et al. (2009)

Objective	Data set	≤ 1	≤ 10	≤ 100	≤ 600	Av. secs
(1) feasibility	Set_{720}	609	662	680	683	2.5
(2) min tank cleaning	Set_{720}	555	677	707	716	4.1
(3) max av. vacant cap.	Set_{720}	546	672	711	717	5.1

method that is to be used to solve a subproblem numerous times in e.g. column generation or local search.

In order to enable a fair comparison with the results from our heuristic, we ran their algorithm with the three different objectives on the same machine used for our heuristic results and also used the newer version 12.4 of CPLEX, that we use for solving the small LPs when determining the cargo amount to put into each tank. Table 4 therefore gives the same numbers as Table 3, however, for a 4.0GB RAM PC with Intel Core2 Duo, 2.4 GHz processor. Since this updated version is faster than the original version, we have added a column with the smaller time limit of 0.1 second and a column with time limit 250 seconds.

Table 4: Results from updated version of optimal method from Hvattum et al. (2009)

Objective	Data set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	≤ 600	Av. secs
(1) feasibility	Set_{720}	298	659	715	719	720	720	0.767
(2) min tank cleaning	Set_{720}	205	557	694	719	720	720	1.822
(3) max av. vacant cap.	Set_{720}	23	525	690	715	717	717	4.912

From Table 4 we see that the updated version of the algorithm is both faster and better than the old one, as it is now able to solve all instances when using objective (1) or (2). These two methods actually solve all instances within 250 seconds while using objective (3) fails to solve all instances within the time limit of 600 seconds. On average, the no objective method is still fastest. However, now the two other objective functions do not yield better results after 10 seconds as they did before. Instead, using no objective is now best on all accounts regardless of the time limit. Therefore, when comparing with our heuristic, we use the no objective method and denote this updated optimality based method by 'UpdOpt'. We also run this version of the algorithm on the two sets Set_{648} and Set_{486} to enable comparison with our heuristic on these sets. The results from these runs are presented in Table 5 where the time limit has been reduced from 600 to 250 seconds due to the speed up of the algorithm.

Table 5: Results from UpdOpt method

Objective	Data Set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
(1) feasibility	Set_{648}	240	598	644	648	648	0.463
(1) feasibility	Set_{486}	196	463	483	486	486	0.374

From Tables 4 and 5 we note that removing instances with built-in instabilities significantly reduces problem complexity and thereby running times. However, if our method is to apply to bulk operators of all sizes, the running time of the algorithm is still too long for the method to be applicable as a subproblem solver in a column generation or local search based framework.

6.2 Modifying the optimality based method from Hvattum et al. (2009)

As already mentioned, when the focus is as here on feasibility rather than optimality, there is no longer any need to include the cost of tank cleaning nor the constraints for cleaning as we can simply assume that tanks are cleaned if necessary, regardless of cost. Removing the cleaning constraints (7) from the model (1)-(10) means that we can further remove the binary cleaning variables, z_{ct}, to further reduce the model size and thereby also the running times. However, for some reason this observation is neither mentioned nor explored by Hvattum et al. (2009). Instead, we have modified their algorithm by removing all cleaning variables and constraints and denote this updated and modified optimality based method by 'ModOpt'. Table 6 shows the improved results from running this version of the algorithm on the three data sets.

Table 6: Results from ModOpt method

Objective	Data Set	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
(1) feasibility	Set_{720}	310	665	716	719	720	0.726
(1) feasibility	Set_{648}	293	605	645	648	648	0.402
(1) feasibility	Set_{486}	263	467	483	486	486	0.282

From Table 6 we see that removing all variables and constraints related to cleaning, reduced the running times by 5-25%. Even so, these running times are still a bit too long for our purpose.

6.3 Results from the developed heuristic

Now we are ready to present the results from the heuristic described in Section 3. As mentioned in Section 6.1, all heuristic tests have been run on a 4.0GB RAM PC with Intel Core2 Duo, 2.4 GHz processor. In Table 7 we show the summarised results from running the heuristic once on each of the three different data sets using the optimal parameter setting derived in Section 5. Note that for completeness we here include tests on the data sets Set_{720} and Set_{648} even though we know that the heuristic will not perform well on these due to the built-in instabilities in these data instances. Even though the heuristic is developed to quickly assess feasibility and that it has therefore been tuned with a time limit of only 0.2 seconds, we here allow a time usage of 250 seconds to be able to compare with the results from Hvattum et al. (2009). Later, we will lower this time limit and run multiple tests to investigate the stability of the heuristic.

Table 7: Results from a single run of the heuristic

Data Set	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
Set_{720}	289	671	702	712	715	715	1.889
Set_{648}	366	623	637	641	644	644	1.776
Set_{486}	290	475	482	484	485	486	0.426

From Table 7 we see that the heuristic is able to solve all instances in Set_{486} and solves 99% of the instances within 1 second. In fact, 99% of the instances are solved already within 0.4 seconds. As expected, the heuristic performs worse on Set_{648} and Set_{720}. The average running time on these sets are relatively high and the instances with built-in instabilities obviously play a major role here as any unsolved instance causes a time usage of 250 seconds whereby the average solve time is significantly higher than without these instances. Comparing our results with the ones in Tables 4 and 5, we see that our average running times are longer than the ones from the UpdOpt

and ModOpt methods. However, these averages are, as just mentioned, highly affected by just a few difficult instances. Looking instead at the distribution of time usage, we see that our heuristic performs much better than both the UpdOpt method and the ModOpt method when running time is limited. In fact, the time limit has to be at least 10 seconds for the UpdOpt method and the ModOpt method to match or outperform our heuristic.

To get an idea of which instance groups are complicated, Table 8 shows more detailed information on the heuristic results for Set_{486}. Looking just at the average running time can be very misleading since a single instance with a long running time will have a huge impact on the average running time. Therefore, when determining which instance groups are complicated it is more accurate to consider the distribution of running times and consider for which groups the majority of the instances are solved quickly. To assist in the understanding of Table 8, for each instance group we put a '-' in entries where all instances in the group have already been solved at a lower time limit. Thereby, an easy rule of thumb becomes: the more '-' entries in a row, the easier the instance group corresponding to this row was to solve.

Table 8: Detailed results from heuristic on Set_{486}

Subset	#INST	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
TAP	486	290	475	482	484	485	486	0.426
C20	161	153	160	161	-	-	-	0.012
C30	168	95	164	165	166	167	168	1.191
C40	157	42	151	156	157	-	-	0.031
D1	114	64	113	114	-	-	-	0.011
D2	116	74	115	116	-	-	-	0.013
D3	124	76	119	123	124	-	-	0.060
D4	132	76	128	129	130	131	132	1.490
F1	136	79	132	135	136	-	-	0.050
F2	165	94	160	163	164	164	165	1.005
F3	185	117	183	184	184	185	-	0.185
T24/V3	172	149	169	171	172	-	-	0.024
T24/V6	180	63	180	-	-	-	-	0.012
T38/V6	41	24	33	38	39	40	41	4.867
T38/V12	93	54	93	-	-	-	-	0.012

From Table 8 we first note that, rather surprisingly, the cargo count does not give any indication of problem complexity for our heuristic. Instead, we can use the probability distribution for the cargo categories as a clear guide to problem complexity. This makes sense, since the higher the D category, the more category 3 cargoes there are, and hence, the more conflicts there are, resulting in unutilised neighboring capacity. With regards to the minimum/maximum capacity utilisation, i.e. the F parameter, we see no pattern with respect to complexity. This actually makes sense, since a lower value for F means that the ship is not very crowded, i.e. requirements for capacity utilisation are low, but on the other hand the ship is rarely emptied completely and so we cannot decompose into smaller and easier subproblems. There is a weak trend suggesting that instances for the T38 ship are more difficult than instances for the T24 ship. However, a much more important factor is the cargo sizes. The smaller they are, the more cargoes will be on board simultaneously, and hence, the more complex the combinatorial puzzle will be.

We also note from analysing the unsolved instances, that the majority of these derive from an inability to reserve sufficient capacity for each cargo as opposed to ensuring ship stability. In fact, instability issues account for less than 1% of the unsolved instances, suggesting that the procedure for creating stable allocations when selecting tanks, works very well. Since this method is generic enough to handle instances with more than one stability dimension, i.e. consider other dimensions than roll, this also suggests that the heuristic will work well on instances with other stability dimensions included.

Allowing long enough running times, the heuristic can solve all instances in Set_{486}. However, the heuristic is neither developed for nor tuned to long running times. Therefore, we want to get an idea of the randomness of the heuristic performance with smaller time limits. To explore this, we have run the heuristic five times on Set_{486} with a time limit of 0.2 seconds. In Table 9 we report the minimum, average and maximum number of solved instances over the five runs. We also state the standard deviation of the number of solved instances. Note though, that without knowing the true probability distribution, this is a sample deviation assuming equal probability for all outcomes. Therefore, this number is most likely higher than the actual standard deviation where outliers would contribute with less weight. Even so, we note from Table 9 that overall performance of the heuristic is quite stable.

Table 9: Analysis of heuristic performance over five runs with time limit 0.2 seconds

Data Set	Minimum	Average	Maximum	Std.Dev.
Set_{486}	478	480	481	1.1

6.4 Devising a hybrid solution method

So far we have improved the method from Hvattum et al. (2009) and created a heuristic solution method and presented results for both these methods. When comparing these results we see that the heuristic is much faster than the ModOpt method on the majority of instances. However, the heuristic struggles with just a few instances which results in an overall average running time slightly higher than that of the ModOpt method. In Table 10 we compare these two methods for running times below 5 seconds.

Table 10: Comparing the ModOpt method with the heuristic on Set_{486} with time limit 5 seconds

Algorithm	≤ 0.01	≤ 0.025	≤ 0.05	≤ 0.1	≤ 0.5	≤ 1	≤ 5	Av. secs
ModOpt	0	0	107	263	447	467	482	0.229
Heuristic	290	463	471	475	481	482	484	0.048

From Table 10 it is clear that the heuristic is much faster than the ModOpt method on the vast majority of instances. A thorough analysis of the results from these two methods shows that they complement each other quite nicely as the two instances that cause the heuristic trouble are relatively easily solved by the ModOpt method while this instead struggles on lots of instances that are easily solved by the heuristic. This suggests combining the two methods to obtain an even faster method. Since the heuristic is much faster than the ModOpt method on most instances, we combine the two methods by first running the heuristic for a short amount of time and if no feasible solution is found, we run the ModOpt method. However, as described in Section 3, the heuristic splits each problem into smaller subinstances and iteratively tries to solve each of these. This means that even if the heuristic failed to find a feasible solution for a given problem instance, it can easily be the case that it managed to find feasible solutions to one or more of the smaller subinstances. We can utilise this fact to enhance the hybrid method even further. Therefore, we combine the two methods by first running the heuristic for a short amount of time and if no feasible solution is found, we run the ModOpt method, however, only on those subinstances that the heuristic failed to solve. Since we only apply the ModOpt method to smaller subinstances, the computation times will obviously be much shorter than what we saw for the full instances. Table 11 shows the results from running this hybrid method on the three data sets with a heuristic time limit of 0.2 seconds as the heuristic is tuned for.

Table 11: Results from hybrid method

Data Set	≤ 0.01	≤ 0.1	≤ 1	≤ 10	≤ 100	≤ 250	Av. secs
Set_{720}	376	671	701	718	719	720	0.367
Set_{648}	283	616	633	647	648	648	0.123
Set_{486}	295	479	483	486	486	486	0.027

Just as we found from the heuristic test results, Table 11 confirms that the removal of instances with built-in instabilities significantly reduces the average running time. In fact, going from Set_{720} to Set_{648} causes a 66% reduction in average running time, while going from Set_{648} to Set_{486} yields a 78% reduction. Furthermore, we note from Table 11 that Set_{720} requires a time limit of 250 seconds to solve all instances while Set_{648} requires 100 seconds and, finally, all instances in Set_{486} can be solved within only 10 seconds. A closer look at our results shows that these numbers are in fact 185 seconds, 21 seconds and 3 seconds, respectively. We note that with an average running time of just 0.027 second, this hybrid method is certainly efficient enough to be used as a subproblem solver. In the next section we compare this hybrid method to the previously presented methods.

6.5 Comparing algorithms

In order to ease comparison between the different methods, we have gathered the summarised results for Set_{486} in Table 12.

Table 12: Comparing the different algorithms on Set_{486} with time limit 250 seconds

Algorithm	≤ 0.01	≤ 0.05	≤ 0.1	≤ 1	≤ 5	≤ 10	≤ 100	≤ 250	Av. secs
UpdOpt	0	87	196	463	481	483	486	486	0.374
ModOpt	0	107	263	467	482	483	486	486	0.282
Heuristic	290	471	475	482	484	484	485	486	0.426
Hybrid	295	475	479	483	486	486	486	486	0.027

By construction, the heuristic and the hybrid method will perform similarly for the first 0.2 seconds. Thereafter, the hybrid method will utilise CPLEX to solve the remaining instances causing a 94% reduction in the average running time compared to the heuristic that struggles with two instances which have a huge impact on the average time. Note that if the heuristic had been used continuously so that all locked cargoes were also placed in a stable manner, the heuristic performance would most likely improve. Even so, the heuristic clearly outperforms the two optimality based methods as long as running time is below 10 seconds. After 10 seconds the two optimality based methods are better at solving the last two instances that cause the heuristic trouble. Thereby, their average running times are better than that for the heuristic. Since the hybrid method combines the best of the heuristic and the optimality based methods, it outperforms them all. In fact, no matter the allowed time limit, no other method solves more instances than the hybrid method and its average running time is as much as 93% lower than the updated version of the method originally presented by Hvattum et al. (2009) and 90% lower than our modified version of this method.

7 Concluding Remarks

In this paper we have considered the Tank Allocation Problem in bulk shipping from a tactical perspective where the main objective is to quickly assess feasibility of a given ship route rather than finding an optimal tank allocation. We have developed a heuristic for efficiently solving this problem and computational results show that it can solve 99% of the considered feasible instances within 0.4 seconds and all of them if allowed longer time. We have also modified an optimality based method presented in Hvattum et al. (2009) and thereby improved their results. The heuristic struggles on two instances causing an overall longer average running time than found with this modified optimality based method. Looking instead at the distribution of time usage, we see that when running time is below 10 seconds, our heuristic clearly outperforms the modified optimality based method by consistently solving more instances. This observation motivated the construction of a hybrid method that first runs the heuristic for 0.2 seconds and if no feasible solution is found, then runs the modified optimality based method on the parts of the instance that the heuristic has not solved. Computational results shows that on the considered instances the hybrid method cuts between 90% and 94% of average running times compared to the three other methods and consistently solves more instances than the other methods within any given time limit. In fact, the average running time for the hybrid method is just 0.027 seconds which is fast enough to facilitate the inclusion of the tank allocation aspect into the routing and scheduling phase.

Since the shipping industry operates in a both dynamic and stochastic environment, it is also worth mentioning that, as discussed in Section 3.3, our heuristic can be expected to generate solutions that are generally more robust to changes than solutions from the method by Hvattum et al. (2009). This robustness is derived from the fact that the heuristic seeks to allocate each individual cargo in a stable manner whereby the stability of the ship does not rely on cargoes to outbalance each other and hence any cargo can in principle be removed from the route without causing instability for the remaining route.

It should also be mentioned that the heuristic described here is flexible enough to incorporate operational considerations such as ballast tanks and moving cargoes between tanks after allocation (i.e. solving the allocation problem for a single set of cargoes on a leg). Finally, it would be interesting to extend our heuristic to allow flexible cargo sizes since this is often used when shipping liquid products. Likewise, it would be interesting to explore the effect of integrating our hybrid method as a subproblem solution method in a procedure for solving the full routing and scheduling problem with the tank allocation aspect included. Both these extensions are left as promising ideas for further research.

Acknowledgements

The research presented in this paper has been partly funded by The Danish Maritime Fund and we gratefully acknowledge their financial support. This research has also been partially supported by the European Union Seventh Framework Programme (FP7-PEOPLE-2009-IRSES) under grant agreement n° 246647 and from the New Zealand Government as part of the OptALI project.

References

D. Barbucha and W. Filipowicz. Segregated storage problem in maritime transportation. In *IFAC Transportation Systems*, pages 557 – 561, 1997.

M.E. Coccola and C.A. Mendez. Logistics management in maritime transporation systems. *Chemical Engineering Transactions*, 32:1291 – 1296, 2013.

F. Cornillier, G. Laporte, F.F. Boctor, and J. Renaud. The petrol station replenishment problem with time windows. *Computers & Operations Research*, 36:919 – 935, 2009.

U. Derigs, J. Gottlieb, J. Kalkoff, M. Piesche, F. Rothlauf, and U. Vogel. Vehicle routing with compartments: applications, modelling and heuristics. *OR Spectrum*, 33:885 – 914, 2011.

K. Fagerholt and M. Christiansen. A combined ship scheduling and allocation problem. *Journal of the Operational Research Society*, 51:834 – 842, 2000.

L.M. Hvattum, K. Fagerholt, and V.A. Armentano. Tank allocation problems in maritime bulk shipping. *Computers & Operations Research*, 36(11):3051–3060, 2009.

A.S. Jetlund and I.A. Karimi. Improving the logistics of multi-compartment chemical tankers. *Computers and Chemical Engineering*, 28:1267 – 1283, 2004.

K. Kobayashi and M. Kubo. Optimization of oil tanker schedules by decomposition, column generation, and time-space network techniques. *Japan Journal of Industrial and Applied Mathematics*, 27(1):161–173, 2010.

G.L. Martin, S.U. Randhawa, and E.D. McDowell. Computerized container-ship load planning: A methodology and evaluation. *Computers & Industrial Engineering*, 14(4):429–440, 1988.

K.-H. Neo, H.-C. Oh, and I.A. Karimi. Routing and cargo allocation planning of a parcel tanker. In *16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering*, pages 1985 – 1990, 2006.

H.-C. Oh and I.A. Karimi. Routing and scheduling of parcel tankers: a novel solution approach. In A. Bruzzone, F. Longo, Y. Merkuriev, G. Mirabello, and M.A. Piera, editors, *The 11th International Workshop on Harbor Maritime Multimodal Logistics Modeling and Simulation*, pages 98 – 103, September 2008.

P. Schaus, J.-C. Regin, R. Van Schaeren, W. Dullaert, and B. Raa. Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem. In M. Milano, editor, *Constraint Programming 2012*, volume 7514 of *Lecture Notes in Computer Science*, pages 815 – 822. Springer, 2012.

UNCTAD. Review of maritime transport 2012. `http://unctad.org/en/PublicationsLibrary/rmt2012_en.pdf`, November 2012.

G.A. Vouros, T. Panayiotopoulos, and C.D. Spyropoulos. A framework for developing expert loading system for product carriers. *Expert Systems With Applications*, 10(1):113 – 126, 1996.

I.D Wilson and P.A Roach. Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, 51(11):1248–1255, 2000.

X. Wu, H.-C. Oh, I.A. Karimi, M. Goh, and R. de Souza. Tops: Advanced decision support system for port and maritime chemical logistics. *The Asian Journal of Shipping and Logistics*, 27(1): 143 – 156, 2011.