On the Complexity of Container Stowage Planning Problems

Kevin Tierney[a,c], Dario Pacino[a,b], Rune Møller Jensen[a]

[a] *IT University of Copenhagen*
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
[b] *Technical University of Denmark, Department of Transport*
Bygningstorvet 116B, DK-2800 Kgs. Lyngby, Denmark
[c] *University of Paderborn, Department of Business Information Systems*
Warburger Straße 100, 33098 Paderborn, Germany

Abstract

The optimization of container ship and depot operations embeds the k-shift problem, in which containers must be stowed in stacks such that at most k containers must be removed in order to reach containers below them. We first solve an open problem introduced by Avriel et al. (2000) by showing that changing from uncapacitated to capacitated stacks reduces the complexity of this problem from NP-complete to polynomial. We then examine the complexity of the current state-of-the-art abstraction of container ship stowage planning, wherein containers and slots are grouped together. To do this, we define the hatch overstow problem, in which a set of containers are placed on top of the hatches of a container ship such that the number of containers that are stowed on hatches that must be accessed is minimized. We show that this problem is NP-complete by a reduction from the set-covering problem, which means that even abstract formulation of container ship stowage planning is intractable.

Keywords: container stowage, overstowage, computational complexity, liner shipping

1. Introduction

Containerization is an important driver of the global economy, and the container has become a mainstay of world-wide trade. Other than a short period of decline in 2009, the number of containers shipped by the world's shipping lines on container vessels has been steadily increasing for the past several decades [1].

Containers are large metal boxes used for transporting goods, and they are constructed such that they can be stored space efficiently directly on top of each other in stacks. Containers are almost always stored in this way, both in stationary storage areas, such as depots and port terminal yards, and moving storage areas, such as bays of container ships. Another characteristic of these storage areas is that containers arrive and depart at discrete points in time. For a typical depot and yard, trucks load and unload containers daily, while containers stowed in bays of ships are loaded and unloaded at different ports. The order in which containers enter and exit ports and vessels is known well in advance of their transportation, as schedules for container ships are generally planned months or even years ahead of time by shipping lines. This, and the fact that stacks only can be accessed from the top, complicates the decision of where to stow containers in a storage area. We call this general discrete optimization problem *stowage planning*, and it is hard because containers to be retrieved from the storage area must be at or near the top positions of stacks.

If a container must be removed from a stack in order to retrieve a container underneath it, the container being removed is said to be *overstowing* the container being retrieved. The process of removing an overstowing container in order to retrieve a container beneath it is called *shifting*. Thus, the goal of stowage planning is to minimize shifting or, equivalently, to minimize overstowage. This is particularly important in bays of container ships, because shifting requires that the overstowing container is moved from the ship to the terminal yard and back, which is very expensive.

Email addresses: kevin.tierney@upb.de (Kevin Tierney), darpa@transport.dtu.dk (Dario Pacino), rmj@itu.dk (Rune Møller Jensen)

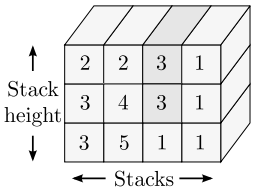

Figure 1: Containers in stacks labeled with the time point of their removal from the stack, with dark gray containers in overstowing positions.

Despite the key economic importance of container stowage problems, little is known about the combinatorial complexity of these problems. To this end, this article has two contributions. First, we solve an open problem posed by Avriel et al. (2000) [2] by showing that a change from uncapacitated to capacitated stacks in their k-shift problem reduces its complexity from NP-complete to polynomial. We do this by providing an algorithm that for any choice of the number of stacks and stack capacities solves the problem in polynomial time.

Second, we examine the complexity of the most successful current abstraction of the stowage planning problem for container ships, in which containers that must be loaded are assigned to vessel bays instead of specific slots (e.g., [3, 4, 5]). In this way, overstowage between containers in individual stacks is ignored, but overstowage can still occur in this model. A metal lid called a hatch separates a bay into two storage areas, one below deck and one on deck. Since the containers stored on deck rest on the hatch, and the hatch must be removed to reach containers below deck, overstowage between containers stored over and under a hatch must be avoided (see Figure 3). In this particular problem, overstowage can also happen during the loading operations if the containers are to be stowed below deck under a hatch cover that already has containers. We formally define this problem as the *hatch overstow problem* (HOP), and show that it is NP-complete. Our result thus shows the necessity of mixed-integer programming and other rich optimization frameworks, as the problem cannot be defined in a purely linear model, unless P = NP.

2. The Capacitated k-Shift Problem

The Capacitated k-Shift Problem (CkSP) is a decision problem that asks whether a set of containers that must be stored and retrieved at a fixed set of discrete time points can be stowed with less than k shifts in a fixed number of stacks with limited capacity. A shift is defined as the removal and replacement of a single container that overstows another container. Multiple shifts may be necessary to reach an overstowed container. Figure 1 shows containers in stacks at a single time point in which two containers, shown in dark gray, are overstowing the container in the bottom row in the third stack from the left. The lower container must be removed at the first time point, and in order to access this container the two containers leaving the stacks at time point 3 must be first removed. Thus, in order to remove the bottom container two shifts occur, one for each overstowing container in the middle and upper positions.

The CkSP is a generalization of the uncapacitated k-Shift problem introduced by Avriel et al. (2000) [2]. Formally, an instance of the CkSP is a tuple $\langle n, C, in, out, S, m, k\rangle$, where n is the number of time points, C is a finite set of containers, $in(c) \in \{1, \ldots, n\}$ ($out(c) \in \{1, \ldots, n\}$) is the time point that container c must be stowed in (retrieved from) one of the stacks, S is a finite set of stacks, $m \in \mathbb{N}$ is the maximum number of containers that each stack can hold at any time, and k is the maximum number of allowed shifts. Note that we avoid detailed models of stacks and vessel stability constraints because, first, these do not change the theoretical difficulty of the problem and, second, the CkSP is a general problem applying to both ports and vessels, meaning stability constraints are not always applicable.

The question is whether the containers can be assigned to the stacks such that at most k shifts are required to retrieve them. Formally, is there an assignment $A : C \to S$ that is within the stack capacity (i.e., $\forall t \in \{1 \ldots n\}, s \in S \,.\, |\{\, c \mid A(c) = s, in(c) \leq t < out(c)\}| \leq m$) that requires at most k shifts (i.e., $|\{w \in C \mid \exists v, A(v) = A(w) \wedge in(v) < in(w) < out(v) \;\wedge\; in(w) < out(v) < out(w)\}| \leq k$)?

Example 1. Consider the CkSP depicted in Figure 2, in which $C = \{c_1, \ldots, c_{13}\}$, $in(c_1, \ldots, c_4) = 1$, $in(c_5, c_6) = 2$, $in(c_7, \ldots, c_{10}) = 3$, $in(c_{11}) = 4$, $in(c_{12}, c_{13}) = 5$, $out(c_1, \ldots, c_4) = 3$, $out(c_5) = 4$, $out(c_6) = 6$, $out(c_7, \ldots, c_{10}) = 5$, $out(c_{11}, c_{12}, c_{13}) = 6$, $S = \{s_1, s_2\}$, $m = 3$. The answer to this CkSP is "yes" for all $k \geq 3$.

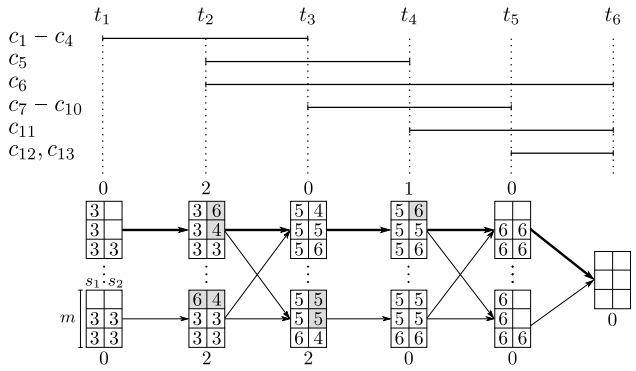

Figure 2: A CkSP instance with $m = 3$, $|S| = 2$ and $|C| = 13$. The instance is a "yes"-instance for any $k \geq 3$. Possible configurations are shown at each time step with the discharge time of the container in each slot. Gray slots contain containers that must be shifted, and each configuration is labeled with the number of containers that will be shifted at that time point. A path through the configurations represents a complete stowage plan, and an optimal plan is shown with bold arcs.

The uncapacitated version of the CkSP where $m = \infty$ has been shown to be NP-complete for $|S| \geq 4$ by a reduction from the coloring of overlap graphs [2] and is known to be polynomial for $|S| < 4$ [6, 7]. For $m = 1$, the CkSP is solvable in time polynomial in C using a minimal cost flow formulation for interval scheduling on identical machines [8]. However, even for $m = 2$, it is not known whether the CkSP can be solved with a polynomial time algorithm. We show that a polynomial time algorithm exists to solve the CkSP for any choice of $|S|$ and m.

We define a *configuration* as a function $q : \{1, \ldots, m\} \times \{1, \ldots, |S|\} \to \{0, \ldots, n\}$ assigning slots to discharge times, in which 0 represents an empty slot and an integer between 1 and n is the discharge time. In other words, when a container c is loaded into a slot in a particular stack, we must only make note of its discharge time ($out(c)$) in the configuration, yielding at most n^σ configurations at any time point, where $\sigma = m|S|$ represents the total number of slots for stowing containers. Let $\perp$ represent an empty configuration. Given a configuration q and a time point $t \geq 1$, let $Q(q, t)$ be the set of configurations that q may *transition* to at time t. We allow a transition between configurations q and q' at time t only when it is possible to re-stow the stacks after unloading containers (including shifted containers) such that each slot in q with a container that has a discharge time later than t and is not overstowing a container being discharged at t, has the same slot assignment in q'. Transitions signify configurations that could logically follow one another, as well as ensure that obviously non-optimal move sequences (such as unloading a number of non-overstowing containers), are not considered. Additionally, $Q(\perp, t)$ provides all of the possible configurations at time t, with $Q(\perp, 1)$ representing the initial (empty) configuration. We note that the total number of time points is bounded above by the number of containers, because time points at which no container is loaded or unloaded can be discarded.

We will now give an algorithm that exploits this polynomially bounded number of configurations to solve the CkSP in polynomial time for any choice of m and S. The minimal number of shifts is given by

$$k^* = \min_{q \in Q(\perp, 1)} \mathit{CkSP\text{-}DP}(q, 1),$$

$$\mathit{CkSP\text{-}DP}(q, t) = \begin{cases} 0 & \text{if } t = n, \\ \mathit{shifts}(q, t) + \min\limits_{q' \in Q(q,t)} \{\mathit{CkSP\text{-}DP}(q', t+1)\} & \mathit{otherwise}. \end{cases}$$

The function $\mathit{CkSP\text{-}DP}(q, t)$ represents the minimum number of shifts necessary for unloading containers in configuration q from time t to time n, where $\mathit{shifts}(q, t)$ returns the number of shifts for the current configuration at time t, and $Q(q, t)$ is as previously defined.

Given a configuration q at time t, our dynamic program computes the number of shifts q requires at time t, followed by the minimum cost transition to a state q' at time $t + 1$. When $t = n$, all containers are discharged and no shifts are required, thus the cost of all states when $t = n$ is 0.

We return a "yes" answer to the CkSP if $k^* \leq k$ and "no" otherwise. Figure 2 shows some of the states of the dynamic program as it solves the problem given in Example 1. Each state represents a configuration and shows the discharge times of the containers loaded in each slot. We do not show all of the possible states, because even in this small example the number of states grows rapidly. We show "interesting" states rather than exhaustively enumerate all states, as many states are simply mirrored versions (left-right or top-bottom) of other states. An arc connects configuration q at time t to configuration q' at time $t+1$ if $q' \in Q(q,t)$, meaning there is a transition between the two configurations. A path through the states represents an assignment of containers to stacks, and the total number of shifts is given by the sum of the number of shifts in each state (shown above and below each state).

We first show that determining the configurations that can be transitioned to, $Q(q,t)$, takes polynomial time to compute. We then leverage these results to show that computing k^* takes polynomial time, and thus, solving the CkSP takes polynomial time.

Lemma 1. *Given a configuration q and a time t, computing $Q(q,t+1)$ has complexity $O(\sigma n^\sigma)$.*

Proof. First, note that there are n^σ possible configurations in total since each slot can be assigned any discharge time. We must determine whether a transition from q exists for each of the possible configurations q' at time $t+1$. This check involves ensuring that all containers not leaving or entering the stacks at time $t+1$ in q and q' are in the same positions. Note that shifted containers are considered as leaving and re-entering the stacks, meaning they are allowed to change positions between q and q'. Checking configurations q and q' takes time $O(\sigma)$, since each slot in each stack is examined in constant time. We perform this check between q and all n^σ configurations, giving a total time of $O(\sigma n^\sigma)$. □

Theorem 1. $k^* = \min_{q \in Q(\perp,1)} \mathit{CkSP\text{-}DP}(q,1)$ *can be computed in polynomial time for any choice of m and $|S|$.*

Proof. The minimal number of shifts, k^*, is computed through a dynamic programming procedure which investigates $O(n^{\sigma+1})$ different states, since there are n time steps and n^σ possible configurations at each step. The function $\mathit{shifts}(q,t)$ computes the number of overstowages at each state, and is clearly polynomial, as it involves looking at each stack in q at time t and counting the number of shifts required. Thus, since $\mathit{shifts}(q,t)$ takes polynomial time, and by Lemma 1, processing each state takes polynomial time for a fixed m and S, resulting in a runtime of $O(\sigma n^{\sigma+1})$.

□

Lemma 2. *$\mathit{CkSP\text{-}DP}(q,t)$ returns the minimal number of shifts for any configuration q and time t.*

Proof. We use a proof by induction on n. In the base case, when $t = n$, $\mathit{CkSP\text{-}DP}(q,n) = 0, \forall q \in Q(\perp, n)$, since all containers are unloaded at time n. For the inductive step, assume that all configurations at time t have been assigned the minimal number of shifts, i.e. $\mathit{CkSP\text{-}DP}(q,t)$ returns the minimum number of shifts for any $q \in Q(\perp,t)$, and the dynamic program must compute $\mathit{CkSP\text{-}DP}(q',t-1)$ for all $q' \in Q(\perp,t-1)$. The minimal number of shifts for $\mathit{CkSP\text{-}DP}(q',t-1)$ is therefore $\mathit{shifts}(q',t-1) + \min_{q \in Q(q',t-1)} \{\mathit{CkSP\text{-}DP}(q,t)\}$, because in order to unload containers in state q' at time $t-1$, $\mathit{shifts}(q',t-1)$ shifts must be performed, and the latter part of the term holds by the inductive hypothesis. □

Theorem 2. $k^* = \min_{q \in Q(\perp,1)} \mathit{CkSP\text{-}DP}(q,1)$ *is the minimum number of shifts.*

Proof. By Lemma 2, $\mathit{CkSP\text{-}DP}(q,t)$ returns the minimal number of shifts for any q and t. Since k^* takes the minimum number of shifts computed by $\mathit{CkSP\text{-}DP}(q,t)$ on each of the initial configurations, k^* must equal the minimum number of shifts. □

Note that the presented algorithm proves that also the optimization version of the problem is in P. This also applies to a version of the CkSP where each stack has its own capacity. This can be easily achieved by adding dummy containers which have a fixed position for all configurations.

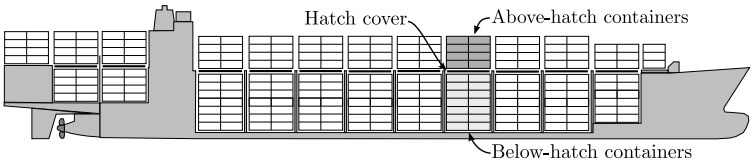

Figure 3: A container ship with hatches separating above-deck (dark gray containers) and below-deck (light gray containers) storage areas.

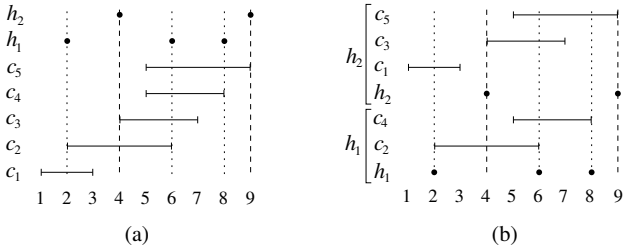

Figure 4: (a) An HOP instance with $m = 2$ and $k = 1$. (b) an assignment of containers to hatches with only one hatch overstow (for h_1) showing that it is a "yes"-instance.

3. The Hatch Overstow Problem

The Hatch Overstow Problem (HOP) is a decision problem that asks whether a set of containers that must be loaded and discharged in a set of numbered ports, visited in order, can be stowed on a set of hatch covers without causing more than k hatch overstows when the hatches may have to be removed in some ports to access some other containers stowed below them. Hatch covers are arranged sequentially across the deck of a container ship. They do not overlap each other and span the width of the ship. Figure 3 shows a cross-section of a container ship with hatches to allow containers to be stored above and below deck. Notice how the removal of any container below a hatch necessitates the removal of all containers resting on the hatch.

Formally, an instance of the HOP is a tuple $\langle C, in, out, H, r, m, k\rangle$, where C is a finite set of containers to stow over the hatches, $in(c) \in \mathbb{N}$ ($out(c) \in \mathbb{N}$) is the port that container c must be loaded to (unloaded from) one of the hatches, H is a finite set of hatches, $r(h) \in 2^{\mathbb{N}}$ is the set of ports where hatch h must be removed, $m \in \mathbb{N}$ is the maximum number of containers that each hatch can hold at any time, and $k \in \{0, \ldots, |C|\}$ is the maximum number of hatch overstows.

The question is whether the containers can be assigned to the hatches without causing more than k hatch overstows. Formally, is there an assignment $A : C \to H$ that is within the hatch capacity (i.e., $\forall t \in \mathbb{N}, h \in H \,.\, \big|\{\, c \mid A(c) = h, in(c) \le t < out(c)\}\big| \le m$) and has at most k hatch overstows (i.e., $\big|\{\, c \mid \exists p \in r(A(c)) \,.\, in(c) < p < out(c)\}\big| \le k$).

Example 2. Consider an HOP with $C = \{c_1, c_2, c_3, c_4, c_5\}$, $in(c_1) = 1$, $in(c_2) = 2$, $in(c_3) = 4$, $in(c_4) = 5$, $in(c_5) = 5$, $out(c_1) = 3$, $out(c_2) = 6$, $out(c_3) = 7$, $out(c_4) = 8$, $out(c_5) = 9$, $H = \{h_1, h_2\}$, $r(h_1) = \{2, 6, 8\}$, $r(h_2) = \{4, 9\}$, $m = 2$, and $k = 1$. As depicted in Figure 4, the answer to this HOP is "yes".

We now prove that the HOP is NP-complete by a reduction from the Set Cover Problem (SCP). Recall that an instance of the SCP is a tuple $\langle S, \mathcal{A}, k'\rangle$, where S is a finite set, $\mathcal{A}$ is a collection of non-empty subsets of S, and $k' \in \{1, \ldots, |\mathcal{A}|\}$. The question is whether $\mathcal{A}$ contains a cover for S of size k' or less, i.e. a sub-collection $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \le k'$ such that every element of S belongs to at least one member of $\mathcal{A}$.[1]

Example 3. Consider an SCP with $\mathcal{A} = \{a_1, a_2, a_3\}$ where $a_1 = \{e_1, e_3\}$, $a_2 = \{e_3, e_4\}$, and $a_3 = \{e_1, e_2, e_4\}$, $S = \{e_1, e_2, e_3, e_4\}$, $k' = 2$. Clearly this is a "yes"-instance since S can be covered by $\mathcal{A}' = \{a_1, a_3\}$.

[1]There are slight variations of the SCP in the literature. The one presented here differs from SP5 in [9] by requiring that the subsets of $\mathcal{A}$ are non-empty. It is trivial to show that SCP is NP-complete by a reduction from SP5.

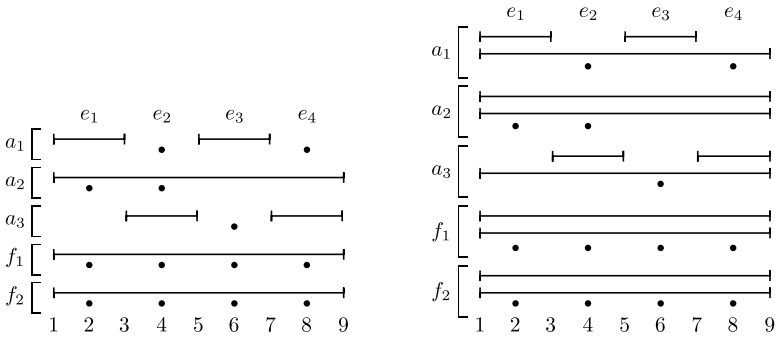

Figure 5: The HOP instance of the SCP defined in Example 3 with both hatch sizes $m = 1$ and $m = 2$. As in Figure 4(b), we show an assignment that proves the HOP to be a "yes"-instance, since there are no more than 3 hatch overstows.

Theorem 3. *HOP is NP-complete.*

Proof. We have HOP $\in$ NP since the assignment A can be used as a certificate that can be checked in polynomial time. We next prove that SCP $\leq_p$ HOP which shows that the HOP is NP-hard. The reduction begins with an instance $\langle\{e_1, \ldots, e_{|S|}\}, \{a_1, \ldots, a_{|\mathcal{A}|}\}, k'\rangle$ of SCP. We shall construct a HOP instance $\langle C, in, out, H, r, m, k\rangle$ that has an assignment with no more than k hatch overstows if and only if the SCP instance has a cover with a size no greater than k'. The HOP instance is constructed by reducing the hatch capacity to one and using the containers to represent the elements in S and using the hatches to represent $\mathcal{A}$. The idea is that a container that belongs to a subset can be assigned to a hatch representing it without causing overstowage. To measure the size of these non-overstowing covers, $|\mathcal{A}|$ blocking containers and k' extra hatches are introduced. Formally, we have a set of containers $C = S \cup \{b_1, \ldots, b_{|\mathcal{A}|}\}$, consisting of containers representing the cover and $|\mathcal{A}|$ blocking containers, where $\langle in(e_i), out(e_i)\rangle = \langle 2i-1, 2i+1\rangle$ for $1 \leq i \leq |S|$ and $\langle in(b_i), out(b_i)\rangle = \langle 1, 2|S| + 1\rangle$ for $1 \leq i \leq |\mathcal{A}|$. Further, $H = \mathcal{A} \cup \{f_1, \ldots, f_{k'}\}$, where $r(a_i) = \{2j \mid e_j \in S \setminus a_i\}$ for $1 \leq i \leq |\mathcal{A}|$ and $r(f_i) = \{2j \mid e_j \in S\}$ for $1 \leq i \leq k'$. Finally, we have $m = 1$ and $k = |\mathcal{A}|$. Clearly, this HOP instance can be constructed in polynomial time.

As an example of the construction, Figure 5a shows the HOP instance of the SCP defined in Example 3, in which the containers (horizontal lines) and hatches are superimposed to show the assignment of containers to hatches. Notice both the containers representing the elements of a_1, a_2 and a_3 (short horizontal lines) and the blocking containers (long horizontal lines). Hatches are listed on the left side, with a_1, a_2 and a_3 corresponding directly to the subsets in $\mathcal{A}$ and f_1 and f_2 being hatches corresponding to k, which in this example is equal to 2. The items of the SCP, e_1 through e_4, are shown across the top of the figure. Each hatch has a removal (shown as a dot) for each item *not* in the hatch's corresponding subset of items.

We must show that this transformation of SCP into HOP is a reduction. First suppose that the SCP has a cover $\mathcal{A}' = \{a'_1, \ldots, a'_n\}$ where $n \leq k'$. For the corresponding HOP, assign each container representing an element in S to a hatch representing a subset in the cover that includes it. This is possible with $m = 1$ because none of these container transports overlap each other. Further, none of these assignments overstow. Then assign $|\mathcal{A}| - n$ blocking containers to the hatches representing subsets that are not included in the cover and assign the remaining n blocking containers to extra hatches. This is possible since no other containers are assigned to these hatches and there are enough extra hatches because $n \leq k'$. Since all the subsets in $\mathcal{A}$ are assumed to be non-empty, we have that all $|\mathcal{A}|$ assignments of blocking containers overstow. The number of overstowing containers, however, is still no greater than k as required.

Conversely, suppose that the HOP has a feasible assignment with $n \leq k$ overstows. Since all blocking containers overstow, we must have $n \geq |\mathcal{A}|$. But since $k = |\mathcal{A}|$, we have $n = k = |\mathcal{A}|$. This means that no element containers overstow, which is only possible if they are assigned to hatches that represent subsets that form a cover $\mathcal{A}'$ of S. The size of this cover can at most be k' because every subset in the cover requires an extra hatch to move a blocking container to and there are only k' of these extra hatches. □

We note that the preceding proof can be applied with only small modifications for any integer value of m, the number of containers that can be stowed on each hatch. Given an arbitrary value of m greater than 1, we create

$\beta = (m-1)(k' + |\mathcal{A}|)$ blocking containers in addition to the k' blocking containers that are part of the original proof. In other words, we add $m-1$ extra containers for each hatch created in the reduction. We leave the non-blocking containers and hatch construction the same as in Theorem 3. For each additional blocking container b'_i for $1 \leq i \leq \beta$, let $\langle in(b'_i), out(b'_i)\rangle = \langle 1, 2|S|+1\rangle$. Further, we assign k in the HOP to be $|\mathcal{A}|+\beta$. This k accounts for the $|\mathcal{A}|$ overstowing containers from the case of $m = 1$, the extra $(m-1)$ containers for each of the $|\mathcal{A}|$ hatches, as well as the $(m-1)$ overstows for each of the hatches $f_1, \ldots, f_{k'}$ in addition to the overstows from the original k' blocking containers.

Figure 5b shows Example 3 with $m = 2$ with the extra blocking containers. Note how each hatch receives an extra blocking container that will always overstow. For higher values of m, more blocking containers are created.

Transforming the SCP into the HOP with any arbitrary m is a reduction for the same reasons as in the case of $m = 1$, except that now we add additional blocking containers to the HOP after assigning the SCP cover and the original blocking containers. The number of overstowing containers does not exceed our redefined k, since all of the original and additional blocking containers overstow, but the element containers do not. Given a feasible assignment to the HOP with $n \leq k$ overstows, we have $n \geq |\mathcal{A}| + \beta$ since all of the blocking containers (original and additional) must overstow. But since $k = |\mathcal{A}| + \beta$, we have $n = k = |\mathcal{A}| + \beta$, meaning that even in the case of an arbitrary value for m, no element containers overstow. This remains only possible if they are assigned to hatches that form a cover of $\mathcal{A}'$ of S.

4. Conclusion

We investigated the complexity of stowage planning problems. We showed that the capacitated k-shift problem is solvable in polynomial time for any choice of stacks and stack capacities, which is an open problem from [2]. We also introduced the hatch overstow problem and showed that it is NP-complete with a reduction from the set covering problem. The complexity of several variations of the CkSP remain unknown, such as when the stack capacity is variable and the number of stacks is greater than the number of discrete time points. Another CkSP variation with unknown complexity has a fixed stack capacity and a variable number of stacks that is greater than three.

5. Acknowledgements

This work was partially funded by Den Danske Maritime Fond under the BAYSTOW project. We thank Thore Husfeldt for his comments on an early version of this work.

References

[1] United Nations Conference on Trade and Development (UNCTAD), Review of maritime transport, 2012.
[2] M. Avriel, M. Penn, N. Shpirer, Container ship stowage problem: Complexity and connection to the coloring of circle graphs, Discrete Applied Mathematics 103 (2000) 271–279.
[3] D. Pacino, A. Delgado, R. Jensen, T. Bebbington, Fast generation of near-optimal plans for eco-efficient stowage of large container vessels, in: Computational Logistics, volume 6971 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2011, pp. 286–301.
[4] J. Kang, Y. Kim, Stowage Planning in Maritime Container Transportation, Journal of the Operations Research society 53 (2002) 415–426.
[5] I. D. Wilson, P. Roach, Container Stowage Planning: A Methodology for Generating Computerised Solutions, Journal of the Operational Research Society 51 (2000) 248–255.
[6] A. Aslidis, Minimizing of overstowage in containership operations, Operations Research 90 (1990) 457–471.
[7] W. Unger, The complexity of coloring circle graphs, in: Proceedings of STACS 92, volume 577 of *Lecture Notes in Computer Science*, Springer, 1992, pp. 389–400.
[8] K. I. Bouzina, H. Emmons, Interval scheduling on identical machines, Journal of Global Optimization 9 (1996) 379–393.
[9] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.