Poznań University of Technology

Heuristics for Long Time Horizon Berth Allocation Problem

Jakub Wawrzyniak, Maciej Drozdowski, Éric Sanlaville,
Xavier Schepler, Jakub Marszałkowski

Research Report RA-06/17

2017

Institute of Computing Science, Piotrowo 2, 60-965 Poznań, Poland

Heuristics for Long Time Horizon Berth Allocation Problem

Jakub Wawrzyniak*, Maciej Drozdowski*, Éric Sanlaville‡,
Xavier Schepler†, Jakub Marszałkowski*

*Institute of Computing Science, Poznań University of Technology, Poland
‡LITIS, Normandy University, UNIHAVRE, Le Havre, France
†LERIA, Université d'Angers, France

Abstract

In this paper berth allocation problem (BAP) for strategic decision making is considered. BAP consists in scheduling ships on a set of berths subject to ready times, ship size constraints, and minimum turnaround time. For the purposes of strategic port capacity planning the BAP must be solved with long time horizons which are hardly ever studied in the current literature. At the strategic level, large uncertainties of ship arrivals and handling times must be dealt with. This calls for use of simulation exploiting fast BAP solving algorithms. A set of heuristics is proposed which can be used for solving big instances of the BAP emerging when time horizons of months and years come into consideration. Performance of the heuristics is analyzed with respect to quality of solutions and runtime. Best methods subject to runtime limitations are proposed.

Keywords: Scheduling; berth allocation problem; metaheuristics; evaluation of heuristics; logistics;

1 Introduction

Maritime container terminals have key importance in the global trade as the transshipment points where the modes of transportation change. There is strong competition between neighboring ports [17, 20]. Therefore, port authorities must consider factors attracting maritime traffic such as vessel turnaround time [18, 20]. Vessel turnaround time is determined by the allocation and sequencing of the ships and the cranes at the berths, availability of the intermediate container storage, throughput of the internal and hinterland connections. Hence, problems like Quay Crane Assignment Problem (QCAP), Quay Crane Scheduling Problem (QCSP), Berth Allocation Problem (BAP) were considered in the literature [1, 2]. In this paper we consider

the Berth Allocation Problem. BAP is defined by a set of ships and berths. Its solution is a schedule of the ships on the berths with the mooring and departure dates. Mean weighted turnaround time is the most frequently used optimality criterion. BAP is usually solved before the QCAP, QCSP, or container storage allocation (yard management), because ships are the most valuable and the least flexible element of the harbor logistics [1]. Consequently, assigning vessels to berths in the port is one of the most critical responsibilities of the port manager.

The BAP has been studied in many papers. On the one hand, most of the papers focus on short term horizons typical of operational and tactical port management. And though sometimes BAP can be solved to optimality, the solutions are given for several days, and consider less than one hundred vessels. On the other hand, in the strategic planning of the terminal evolution, partitioning of quays into berths (dimensioning and layout), the number of quay cranes, the capacities of the container storage and internal modes of transport, can be optimized for maximum overall terminal throughput. Such a strategic planning involves long term horizons of months and years. The berthing space is a potential bottleneck which may limit throughput of a port. Thus, large BAP instances with hundreds and even thousands of ships have to be solved in strategic port capacity studies.

The global maritime traffic, and the share of this traffic for a given port, may be estimated with some accuracy for a few forthcoming months. Yet, with the perspective of years, there is unavoidable uncertainty in such data. Considering the large versatility of possible vessel arrivals, the amounts of shipment, *simulation* studies are inevitable in the strategic planning of the terminal evolution. Since the number of possible traffic development scenarios, and options for 'what if' studies is large, also the number of simulation runs will have to be large. Hence, there is a need for solving large BAP instances in short runtime. Since the BAP is an **NP**-hard optimization problem, it can be solved to optimality only for small instances (and hence very short time horizons) using MILP formulations (see e.g. [8, 15]). For larger instances metaheuristics dominate as solution methods [2, 6, 12]. Since these methods can run minutes and even hours, they cannot be applied in solving a year-long BAP as multiple one week problems, or in the rolling horizon setting. The work presented in this paper is an attempt to provide fast methods capable of solving large BAPs emerging with the planning horizon of months and even years.

Further organization of the paper is as follows. The next section presents related literature on the Berth Allocation Problem. In Section 3 BAP is formally defined. Section 4 is dedicated to greedy algorithms solving the problem, while more advanced heuristics are introduced in Section 5. These algorithms are evaluated in Section 6. The last section is dedicated to conclusions.

2 Related work

The number of papers devoted to the BAP exceeds one hundred. Hence, it is not possible to report on all of them here. We will focus on introducing basic concepts of the BAP and the difference between strategic, tactical and operational BAP instead. For an extended survey see Bierwirth and Meisel [1, 2]. Variants of the BAP are defined by a broad set of spatial, temporal, performance attributes describing the quay layout, the vessel service process, or the objective function [1, 2]. One of the main determinants of the BAP is the quay partitioning model. It may be discrete, continuous, or hybrid. The discrete model assumes that a quay is partitioned into fixed berths, and one berth may accommodate one vessel at a time. The continuous model assumes that vessels may be assigned to moor at arbitrary positions along a quay. The hybrid model assumes a fixed quay partitioning, but small vessels may share one berth, while large vessels may occupy more than one berth. The berths occupied by one ship at the time will be called segments. The optimality criteria try to capture the goals of the port and terminal managers, or line shippers. Hence, they are quite numerous and sometimes contradictory. The mean weighted turnaround time is used most often. The turnaround time is the time a vessel spends waiting and mooring. Thus, mean weighted turnaround time is equivalent to the mean weighted flow time frequently referred to in the scheduling theory. The popularity of this criterion mirrors the high competition between ports willing to attract and keep the shipping lines. The second most frequently used criterion is the weighted sum of vessel tardiness. Other criteria minimize, e.g., the cost of container handling by the terminal.

2.1 Strategic BAP

In the strategic level of planning the choice of the liner services guides investments in new facilities extending capacity of the port. Hendriks et al. [10] study assigning time windows and terminals to cyclically calling vessels with the goal of minimizing quay crane workload over time and minimizing inter-terminal container transport. Imai et al. [11] define the strategic BAP as selecting maritime shipping lines for the terminal and assigning them cyclic berth templates (i.e. temporal and spatial positions) with the objective of minimizing port operation costs and observing constraints of the shipping lines. They admit that ship data may be imprecise for this long-term planning. The cranes will be scheduled and yard storage assigned at the tactical level. If the strategic decision turns out to be infeasible at the tactical level, then the terminal operator may either invest into expanding crane and storage capacity or change the strategic decision. Let us observe that the uncertainty of ship arrivals and capacity of handling resources is so high, that the deterministic cyclic scheduling is just a model of reality.

Alternatively, extensive simulations based on fast BAP solving methods can be applied. The algorithms considered in this paper are instrumental in this process, allowing for early verifying feasibility of the decisions made in the strategic stage. What is more, these algorithms can be applied at even earlier stages when a container terminal is designed.

2.2 Tactical BAP

In the tactical BAP time horizons are studied, typically one or two weeks. The input data are still highly uncertain. The studies on the maritime and port traffic demonstrate that over 40% of vessels arrive one or more days behind the schedule [19]. But still, since some decisions must be taken, deterministic version of the BAP is considered with the assumption that vessel schedules will be adjusted at the operational level. Since the unloading and loading times depend on the resource allocation, berth allocation decisions are often coupled with quay crane or container storage allocation.

Most papers assuming the tactical level planning consider the discrete BAP. In Giallombardo et al. [7], the BAP and the quay crane allocation (QCAP) are solved simultaneously, thus leading to a bicriterion optimization. Hendriks et al. [9] consider the BAP and the storage allocation problem, the objective is minimization of the distance to the storage area. Zhen et al. [22] analyze also the storage allocation problem. Again, this is a bicriterion optimization: minimizing the cost of container handling and the deviation from the schedule of the shipping companies. Zhen [21] studies a cyclic model, minimizing the deviation from the initial schedule of the shipping companies, while the processing times follow some random distribution. However, the random variables affect only the objective function. Legato et al. [13] model the processing times by β distributions, according to data from a real terminal. They validate the obtained solutions by simulation. Moorthy et al. [14] consider a cyclic model and minimize several conflicting criteria like mean waiting time and container flow costs. Their goal is the schedule robustness which is achieved by adding time buffers to processing times to cope with uncertainties.

2.3 Operational BAP

In the operational BAP it is assumed that all data, such as arrival dates and processing times of the vessels, are known and fixed. This BAP setting is relevant to short time planning (maximum a few days). The decisions taken at the tactical level may be taken into account. Then, the goal of the operational planning is to stay as close as possible to the initial tactical level schedule. This is aided by applying some schedule stability criteria.

Even though most variants of BAP are **NP**-hard, they can often be solved exactly or nearly exactly for small instance sizes. The best exact approach

so far for the discrete variant is based on the generalized set partitioning model from Buhrkal et al. [4]. It solves the classical instances from the literature of 12 days, up to 60 vessels, and 13 berths in just a few seconds. In [8] instances with up to 10 berths and 50 ships were solved in one hour time limit. Metaheuristics were also used with good success, as the solutions obtained by the tabu method of Cordeau et al. [6] were usually within 1% of the optimum. Lalla-Ruiz et al. [12] use tabu search with path relinking for the operational BAP. They provided near optimal solutions in less than a few seconds to instances with up to 60 vessels, 13 berths. The continuous variant is considered more difficult and metaheuristics are mostly used.

There are fewer works on the hybrid variant, most of them with metaheuristics, and often with specific additional features. In the Schepler thesis [15] (see also [16]), a port planning problem is considered with the BAP as a sub-problem. Instances with up to 7 days of planning, 48 vessels, on several terminals are solved. The model considers also trains, trucks, and inter-terminal transport.

Let us comment on the utility of some simple port throughput models based on quantities of shipped containers. These models are bound to limited accuracy only if the shipped commodity (containers) is treated as a medium continuously divisible in processing time and space. The existing results in the deterministic scheduling theory [3, 5] provide indications that such models may significantly diverge from feasible schedules, because ships are discrete objects. For example, in [5] it has been shown that the ratio of preemptive and nonpreemptive schedule lengths on parallel identical processors is at most 4/3. In terms of the BAP this means that if re-berthing of the ships and suspension of their processing are disallowed (which is usually the case), then the best schedule can be by $\approx 33\%$ longer than an optimistic approximation perceiving work as a divisible medium. In [3] contiguous and non-contiguous parallel task scheduling has been considered. In the BAP setting the problem studied in [3] assumes a hybrid quay partitioning. The non-contiguous assignment of ships to quays has no practical sense here because ships are not divisible along their lengths. However, such an assumption is equivalent to allowing for approximation by the container flows. It has been shown in [3] that already with 4 segments, or with 7 ships, or in schedules 4 time units long such estimations may diverge from feasible schedules. Furthermore, the bound on the difference between such schedule lengths (i.e. the non-contiguous approximation and the contiguous feasible schedule) obtained analytically is 100% of the schedule length, while simulation studies demonstrated that differences greater than 15% are hardly ever found. Hence, to obtain credible results on port capacity, simulation studies based on the actual schedules are necessary and solving the BAP cannot be substituted by simple container flow models.

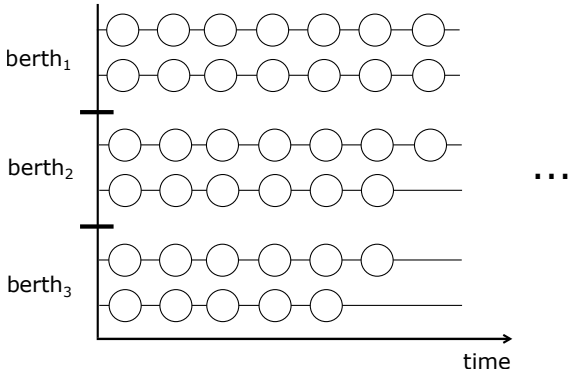


Figure 1: Solution representation

3 Problem formulation

In this paper it is assumed that a set of m berths is given. A berth is defined by its length λ_i, for $i = 1, \ldots, m$. There are n vessels defined by: arrival times r_j, lengths L_j, processing (i.e. unloading and loading) times p_j, and importance w_j, for $j = 1, \ldots, n$. Vessel j can be moored at berth i only if $L_j \leq \lambda_i$. Ship lengths are limited to a small set of the typical ship sizes following from the sizes of locks, channels and ports. In particular, we will consider the popular classes of container ships like Suezmax and (New) Panamax. Importance w_j of ship j represents the value of the ship for a port authority. This can be understood as, e.g., the cost of mooring, or the cost of the cargo.

We assume hybrid quay organization. The quay is divided into discrete berths. Each berth can accommodate at most two ships at the same time. This guarantees that if the total length of two ships does not exceed the berth length, then they are always feasibly schedulable at the berth without a need for re-berthing (moving) the ships. We will represent a solution of the BAP as a set of chains, where each chain is a sequence of ships mooring at certain berth (see Fig. 1). Each berth has two chains, called the left and the right chains. The pair of chains represent the sequence of ships moored at the ends of the berth (the left, and the right, respectively).

The objective function to be minimized is the mean weighted flow time $MWFT = \sum_{j=1}^{n}(c_j - r_j)w_j / \sum_{j=1}^{n} w_j$, where c_j is completion time of handling ship j. According to the notation introduced in [1, 2], our problem can be denoted $hybr|dyn|fix|\sum w(wait + hand)$. In the two following sections algorithms to solve the above problem are proposed.

4 Greedy algorithms

In this section we introduce greedy algorithms for the BAP. They are defined by two elements: the control structure and the sorting rule. The control structure determines when the greedy algorithm comes into action and the range of considered ships. The sorting rule is sequencing the ships according to some criterion. There are 5 variants of the control structures and 12 priority rules. Overall, we take into account 60 greedy algorithms.

4.1 Sorting rules

First-Come, First-Served (FCFS) orders the ships by their arrival times: $r_1 \leq r_2 \leq \ldots \leq r_n$.

Longest Ship First (LSF) and *Shortest Ship First (SSF)*, are based on the lengths of the vessels sequenced according to the descending $L_1 \geq L_2 \geq \ldots \geq L_n$, or the ascending order $L_1 \leq L_2 \leq \ldots \leq L_n$ of lengths, respectively.

Longest Processing Time (LPT) and *Shortest Processing Time (SPT)*, rely on the processing time orders, $p_1 \geq p_2 \geq \ldots \geq p_n$ and $p_1 \leq p_2 \leq \ldots \leq p_n$, respectively.

Largest Area First (LAF) and *Smallest Area First (SAF)*, take into account the area of the ships in time×length space: $p_1L_1 \geq p_2L_2 \geq \ldots \geq p_nL_n$ and $p_1L_1 \leq p_2L_2 \leq \ldots \leq p_nL_n$, respectively.

Weighted Shortest Processing Time (WSPT) is based on the vessel processing times divided by their importances $p_1/w_1 \leq p_2/w_2 \leq \ldots \leq p_n/w_n$.

Random (RND) builds a random sequence of ships and is used as a reference algorithm to check whether other algorithms return useful solutions.

Greatest Importance (GI) relies on the vessel importance: $w_1 \geq w_2 \geq \ldots \geq w_n$.

Greatest Importance - Shortest Processing Time (GISPT) builds the order in two steps: First, it sorts the ships by importance, like the GI rule, and then ships of the same importance are ordered as in the SPT rule.

Shortest Processing Time - Greatest Importance (SPTGI) sorts the vessels by SPT rule first, while ships of the same processing time are ordered by the GI rule.

In all the above rules ties are resolved arbitrarily.

4.2 Control structures

Priority heuristics (Prio)

Priority heuristics build solutions by assigning ships to berths according to increasing dates. Scheduling decisions proceed through ship ready times and

ship service completion times. At each such moment, the first ready ship is chosen from the list defined by the sorting rule. Then, the ship is assigned to the shortest available berth. This means that while there are available ships we will try to assign them to the shortest available berths. If we find a ship that does not fit any available berth, or there are no more free berths at this moment, then we go to the next decision moment. The procedure is repeated until there are no ships to schedule.

List heuristics (List)

This control structure relies strictly on the given sorting rule and the list of ships this rule constructs. While the list is not empty, the first ship is chosen and assigned to the earliest shortest feasible berth. This means that berths available at the earliest time are searched for first, and if there are more than one then the shortest feasible one is chosen. The procedure is repeated until all the ships are assigned to the berths. If the chosen ship j is not ready, then the algorithm *waits* until its ready time r_j. Thus, contrary to the previous control structure, the *list* structure allows that a berth is not used by a ready ship, as the berth waits until the arrival of a ship of the higher priority.

k-Look-ahead heuristics (La k)

The k-look-ahead structure acts like the priority structure, but at the decision points it takes into account also k future ship arrivals and include the k future ships into the set from which the next ship to serve is chosen according to the sorting rule. We assume that $k \in \{2, 5, 10\}$ so there are 3 variants of this control structure. Let us observe that k-look-ahead structure is a generalization of both the priority structure ($k = 0$), and the list structure (k decreases from n to 1).

4.3 Greedy heuristic short-hand notation

Since the number of possible greedy algorithms is quite large, we will be using a short form notation to refer to them. A short name will consist of two parts: an abbreviation of a sorting rule name, followed by the abbreviation of the control structure name. For example, SPTGI-Prio is a method combining sorting rule SPTGI with priority control structure, and FCFS-La2 is 2-look-ahead with FCFS sorting heuristic.

Moreover, performance of a combination of all greedy algorithms will be reported under name *Super Greedy (SG)*. This means, that SG algorithm provides the best solution constructed by any greedy algorithm and has the runtime equal to the sum of the runtimes of all greedy methods.

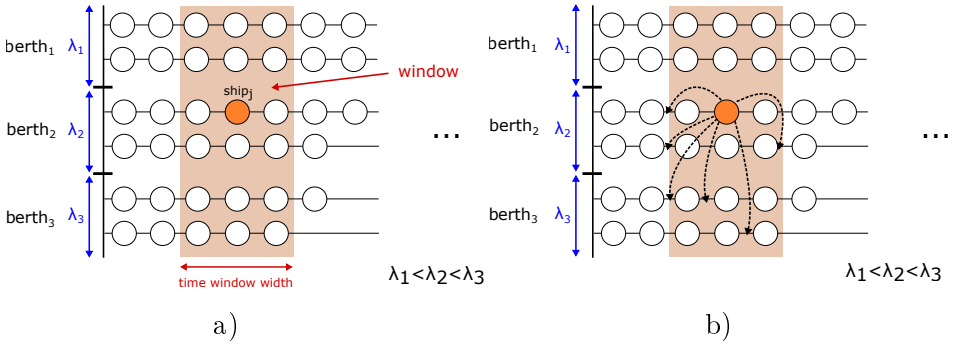


Figure 2: Solution neighborhood in HC. a) Visualization of the constraints, b) feasible moves

5 Local search algorithms

In this section we introduce algorithms based on the local search. These methods start from some initial solution and then attempt to iteratively improve the current solution by modifying it. For all the local search algorithms 1 hour time limit has been imposed.

5.1 Hill-Climber (HC)

The Hill-Climber starts from the best solution S constructed by any of the greedy algorithms described in Section 4. In the first variant, called *HC*, solution neighborhood is defined in the following way. For each vessel, HC tries to move it from the current position in the sequence to a different position (cf. Fig. 2). In principle all feasible positions in all berth chains can be examined. Obviously, it is not allowed to move a ship to a berth that is too short (Fig. 2b). To ensure sufficient time to generate new solutions even for large instances (e.g. for $n \geq 5000$), two constraints on the size of a neighborhood have been introduced: time window of width WL and limit SL on the number of relocated ships. Let st_j denote the start time of handling the ship at position j. A move from the current position j to a different position k is possible only if $st_j - WL \leq st_k \leq st_j + WL$ (cf. Fig. 2a). The time window condition is tested first. The move is executed if the number of the ships considered for relocation so far has not exceed limit SL. All the new solutions are evaluated according to the objective function and the best improving solution S' is chosen. Then, the algorithm is restarted from S'. HC works until there is no improving move or time limit is exceeded. Control parameters have been tuned on a set of randomly generated instances with $n = 1000$ (for the process of test instance generation see Section 6). All pairs of $(WL, SL) \in \{1, 10, 100, 1000\} \times \{1, 10, 100\}$ have been tested. Two measures of algorithm performance have been considered:

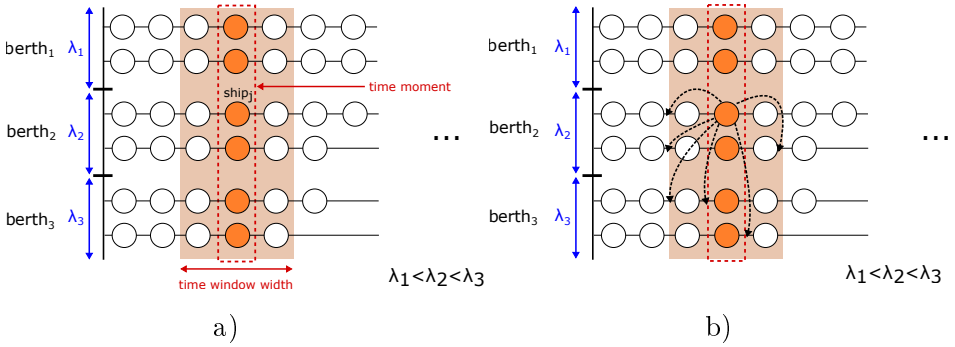


Figure 3: Solution neighborhood in HC-A. a) Visualization of the constraints, b) feasible moves

$MWFT$ improvement from $(WL, SL) = (1, 1)$, and runtime until providing the best solution the algorithm was able to construct. The values of $WL = 10$ and $SL = 10$ have been chosen as giving on average the greatest objective function improvement per unit of time until finding the best solution.

In the second variant, referred to as *HC-A*, a time moment is randomly generated in the existing schedule. All vessels which interval of processing intersects with the chosen time moment are tried for redistribution (cf. Fig. 3a). Similarly to the previous variant, to ensure sufficient time to generate neighbor solutions, two constraints have been introduced: number of attempts NA and time window width WL. The number of attempts NA is checked first to limit the number of times a new time moment is generated. The time window constraint has the same purpose as in the previous variant of the algorithm (cf. Fig. 3b). Control parameters have been tuned on the same set of instances as *HC*. All pairs of $(NA, WL) \in \{10, 100, 1000\} \times \{1, 10, 100, 1000\}$ have been tested. Algorithm performance measures were: $MWFT$ improvement from $(NA, WL) = (10, 1)$, and runtime until providing the best solution the algorithm was able to construct. Values $NA = 10$ and $WL = 10$ have been chosen, because they provided the best results on average.

In the third variant, called *HC-C*, different parts of the existing solution are analyzed separately. Firstly, a time window of a fixed width WL on a random berth, starting at a random time position is selected. Secondly, all vessels in the time window are checked for relocation (cf. Fig 4a). Target time positions st_k for the considered ship have to satisfy the condition $st_j - WL \leq st_k \leq st_j + WL$ (cf. Fig. 4b), where st_j is the time of starting processing the ship in the old schedule. Besides WL, the second control parameter for *HC-C* is the number of attempts NA, i.e. the number of times a new time window can be generated. All control parameters have been tuned using the above mentioned set of instances. The following pairs of (NA, WL) were considered: $(NA, WL) \in \{10, 100, 1000\} \times \{1, 10, 100, 1000\}$. Taking

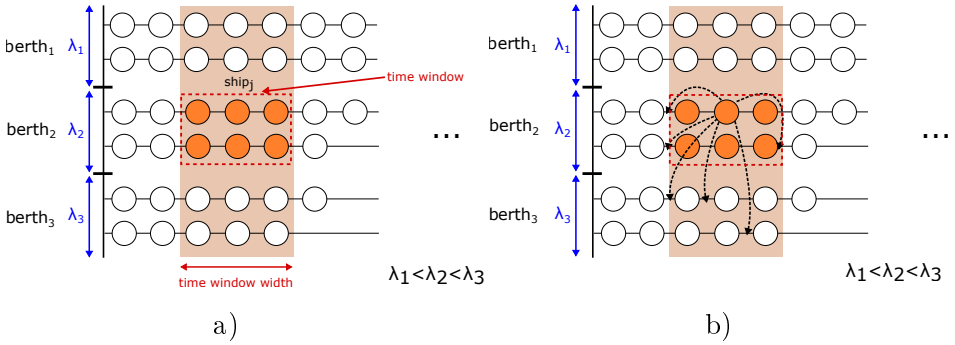


Figure 4: Solution neighborhood in HC-C. a) Visualization of the constraints, b) feasible moves

into account the $MWFT$ improvement from $(NA, WL) = (10, 1)$ and runtime until providing the best solution, values $NA = 10$ and $WL = 10$ have been chosen.

5.2 Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Randomized Adaptive Search Procedure (GRASP) is divided into two parts. In the first part it builds an initial solution. Until all ships are scheduled, the following procedure is repeated: A set of k vessels is chosen from the top of a list constructed according to a given sorting rule. The k ships constitute the so-called restricted candidate list (RCL). Then, one vessel is randomly drawn from the RCL and appended on the earliest shortest feasible berth, i.e. berths are verified in the order of increasing availability times (firstly) and lengths (secondly). When all n ships are scheduled GRASP is restarted. The best solution is always retained. This first part of the algorithm is repeated until the first time limit tl_1 elapses. In the second part HC (Section 5.1) is applied for post-optimization. HC is run until the second time limit tl_2 expires. The selection rules have been limited to GI, GISPT, SPTGI because these sorting rules provided the best solutions in the initial tests against changing instance sizes n, m (cf. Section 6.2, Tab.1). Rules LSF, LAF have been added to diversify the set of GRASP methods and to check if GRASP infrastructure can help weaker sorting rules deliver good solutions.

The split of the overall 1 hour runtime limit into tl_1, tl_2 has been tuned on a set of instances with $n = 1000$. The test values for tl_1 were $\{600, 1200, 1800, 2400, 3000\}$ seconds. The second time limit tl_2 covered the time remaining to the overall 1 hour runtime. On the one hand, it turned out that on average the longer the first part runs, the better solutions are obtained. On the other hand, HC running in the post-optimization stage needs time to search neighborhood of the current solutions at least a few times. This is

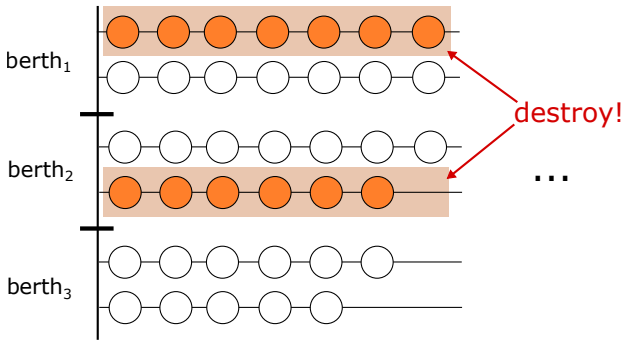


Figure 5: Operation of the ILS-A variant of the Iterated Local Search algorithm

especially severe for $n \geq 5000$. The value of k and selection of HC variant for post-optimization, have been also tuned. The test values for k were $\{2, 5, 10, 20\}$. In the second part of GRASPs, HC ($WL = 10, SL = 10$), HC-A ($NA = 10, WL = 10$), and HC-C ($NA = 10, WL = 10$) have been applied. Tests for all the above mentioned sorting rules have been performed. Two measures of performance have been considered: $MWFT$ improvement from $(k, tl_1) = (2, 600\text{s})$, and runtime until providing the best solution the algorithm was able to construct. As a result, 6 implementations of GRASP will be considered. In order to distinguish them name GRASP followed by the sorting rule abbreviation will be used. For example, GRASP-LSF applied Longest Ship First sorting rule. The 6 GRASP variants will be: GRASP-LSF ($k = 20, tl_1 = 600\text{s}$, HC-A), GRASP-LAF ($k = 20, tl_1 = 600\text{s}$, HC-A), GRASP-GI ($k = 10, tl_1 = 600\text{s}$, HC-A), GRASP-GISPT ($k = 20, tl_1 = 600\text{s}$, HC-A), and GRASP-SPTGI ($k = 10, tl_1 = 600\text{s}$,HC-A). Moreover, a version without HC, referred to as GRASP-3600, will be also examined. In this version GISPT sorting rule has been applied with $tl_1 = 3600\text{s}$ and $k = 10$.

5.3 Iterated Local Search (ILS)

ILS starts from the best solution S returned by any greedy algorithm. Then, ILS iteratively tries to find an improved solution by first destroying and then reconstructing part of S. ILS has two implementations. Both of them stop when time limit is reached.

In the first variant, called *ILS-A*, whole chains in the current solution can be dismantled (Fig. 5). Let $\varepsilon \in [0, 1]$ be the fraction of the number of chains which should be destroyed. For example, for $\varepsilon = 0.2$, ILS-A removes all vessels from 20% of the existing chains. The precise chains to operate upon are chosen randomly with equal probability. A constraint has been

imposed that in each iteration ILS-A has to dismantle at least two chains from at least two different berths. The ships from the dismantled chains are sequenced by a greedy algorithm and appended in this sequence to the chains being reconstructed. Suppose two chains exist on some berth: a new one that is being reconstructed, and an old one which is an unmodified chain remaining from the initial solution S. For some ship j a berth i of length $\lambda_i \geq L_j$ available at the earliest time greater than or equal to r_j is searched for first. If more than one berth meets the above conditions, then the shortest is chosen. The calculation of berth i availability time bat_i takes into account two cases. Let τ be the time when the service of the last ship on the new chain is finished. If no ship is scheduled in the old chain at time τ, then $bat_i = \tau$. Otherwise, bat_i is the time when the service of this ship in the old chain finishes. In other words, if some ship in the old chain overlaps τ, then its completion time is used as bat_i. If both chains are new, then maximum of their completion times is used as bat_i. Assigning ship j to a berth may cause conflict with the schedule of the old chain. Namely, the newly inserted ship j and some ship k scheduled in the old chain in the interval $[bat_i, bat_i + p_j]$ may have lengths such that $L_j + L_k > \lambda_i$. If it is the case, then the ships in the left chain have priority, i.e. the ship at the left end of the berth is scheduled first, while the ship in the right chain is scheduled as the second. The ships remaining in the tail of the old (unmodified) chain are delayed accordingly. All known greedy algorithms are applied to reconstruct the solution and the best solution S' is selected.

In the second variant, referred to as *ILS-C*, ships are removed from the current solution to create „holes" in the existing schedule (see Fig. 6). It is assumed, that holes are created in the chains representing the schedules on the berths, and there can be at most one hole in a chain. Let $\varepsilon \in [0, 1]$ be the fraction of the solution size that must be destroyed, i.e. $n\varepsilon$ ships are removed from the current solution. The ship removal process progresses in three stages. Firstly, chains are selected with equal probability until collecting chains comprising at least $n\varepsilon$ ships. Secondly, the $n\varepsilon$ ships to remove are split randomly between the selected chains with uniform probability. Let z_i be the number of ships to be removed from chain i. Finally, z_i ships are removed from the selected chain i starting at a position chosen with uniform probability along the chain length. In the reconstruction process the ships are ordered by some greedy algorithm and reinserted into holes in the schedule. When some ship j is chosen to be scheduled by some greedy algorithm, then it is assigned to a hole in the earliest shortest feasible berth. This means that the earliest available insertion position in a hole not earlier than r_j is searched for on the berths i such that $\lambda_i \geq L_j$. If there are more than one such position, then the shortest berth is chosen. The availability time of a position in a hole is determined in the following way. Suppose k is the index of the last ship in the chain before a hole. Indices $k + 1, \ldots, k + a$

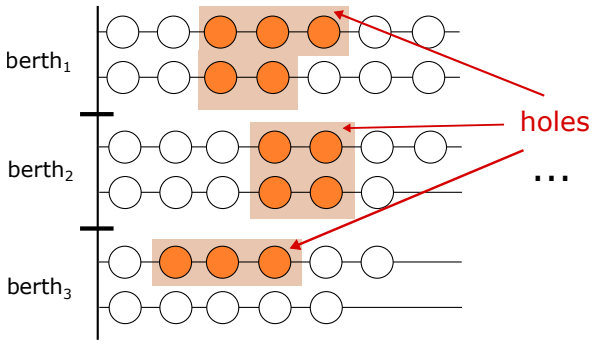


Figure 6: Operation of the ILS-C variant of the Iterated Local Search algorithm

represent ships already inserted in the hole, where $a = 0$ initially. Availability time of the positions in the hole are times $c_k, c_{k+1}, \ldots, c_{k+a}$ when the service of ship $k, k+1, \ldots, k+a$ ends. If some position k' in a hole of chain i is selected, then the ships in the positions later than k' are delayed to allow inserting the considered vessel j. If there is a conflict with the ships on the other chain of the berth, then also they are delayed. Of the two conflicting chains, the left one is always given preference. All greedy algorithms are applied in sequencing the ships to be reinserted into the schedule and the best obtained solution S' is selected. The value of ε parameter has been tuned on a set of randomly generated instances with $n = 1000$. Taking into account performance measured by $MWFT$ improvement and runtime, two versions of ILS will be considered: ILS-A ($\varepsilon = 0.3$) and ILS-C ($\varepsilon = 0.1$).

6 Evaluation of the algorithms

In this section we report on the results of evaluating the algorithms introduced in the preceding sections. We explain the method of test instance generation. Then, the impact of the instance parameter values on the quality of solutions is studied, also the influence of the dispersion of instance parameters on the solutions quality is examined. The algorithms trade the runtime for the solution quality. The efficiency of the algorithms in this trade-off is tested. Finally, we give recommendations on the choice of the best algorithms given runtime limits. The methods of evaluating the heuristics and electing the usable ones can be applied also in other combinatorial optimization problems.

Performance of the algorithms is measured by their runtime and solution quality. The latter will be measured by three indicators: the number of wins,

the number of unique wins, and central tendencies of $MWFT$ (e.g., average, median). The number of wins in a population of instances is the number of instances for which the algorithm returned the best solution. The number of unique wins is the number of instances for which the algorithm returned the best solution exclusively. These two criteria serve as indicators of the algorithm ability to cover a set of instances with the best solutions. The algorithms were implemented in C++, compiled with GNU g++ ver. 4.8.2. The code was executed on a PC-cluster `Eagle` at Poznan Supercomputing and Networking Center, which has 1233 nodes comprising Intel CPU E5-2697 CPUs with 32984 cores running at 2.6 GHz, 64/128/256 GB RAM per node. The operating system was Scientific Linux CERN 6.7 (Carbon). Each algorithm solved 5900 instances overall. Due to space limitations, only selected results are presented.

6.1 Instance generator

Reliable evaluation of the algorithms for long-time horizon BAP requires an extensive number of tests on large problem instances. Using instances from the earlier publications is problematic because the sizes of the instances are small, the data happens to be fragmentary. The instances representing real ports may limit generality of the study because due to the existing correlations in the data some combinations of instance parameter values may be omitted and then the tests may be insufficiently stressful. Therefore, we implemented a generator of test instances. Unless stated to be otherwise parameters values are drawn as follows: $n \sim U[1, 1000]$, $m \sim U[1, 100]$, $r_j \sim U[0, 1000]$, $p_j \sim U[1, 24]$, $w_j \sim U[1, 1000]$, $L_j, \lambda_i \sim U\{200, 215, 290, 305, 400\}$. By $\sim U[a, b]$ we denote that certain parameter is generated from discrete uniform distribution with integer values in range $[a, b]$. The notation $\sim U\{x, \ldots, y\}$ means that the parameter values are chosen with discrete uniform distribution from the set $\{x, \ldots, y\}$. The set of ship lengths L_j represents popular container ship classes like Suezmax and (New) Panamax. Unless stated to be otherwise, each configuration of the tests represents a population of 100 instances. This means that each point in the following figures represents results collected over 100 instances.

In the following the impact of some selected parameter on the algorithm's performance is tested. In such tests the examined parameter has been fixed in all the instances, a range of the tested parameter has been swept, while the remaining parameters have been randomized. For example, in the tests of the impact of the number of ships n, the range of n values has been swept by visiting values $n \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$ one by one. This means that 100 instances have been generated with $n = 2$ while the remaining parameters have been randomly generated as described above. The set of 100 instances with $n = 2$ have been solved by the algorithms to evaluate their performance. Next, values $n = 5, 10, \ldots$ have been examined

in the similar manner. In the examination of the impact of other parameters, the method of instance generation was similar. More details will be provided when discussing a particular test setting.

6.2 Impact of instance parameters on the solution quality

In this section we discuss changes of solution quality with some parameter of the instances when the algorithms are allowed to run until one hour time limit. Thus, the time dimension of algorithm performance is taken out of consideration here. Our quality measure will be the number of wins achieved by the algorithms on a set of instances.

The dependence of the solution quality on the number of ships has been tested by sweeping the following set of ship numbers: $n \in \{2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000\}$, while the remaining instance parameters were randomized as described in Section 6.1. Analogously, the dependence of the solution quality on the number of berths has been tested by sweeping the berth numbers: $m \in \{1, 2, 5, 10, 20, 50, 100\}$ and randomizing the other parameters. These two series of experiments reveal sensitivity of the algorithms to growing instance sizes.

Let us start with a comparison of the algorithm performance by juxtaposing the numbers of instances on which the algorithms win in the tests against changing n and m. These results are collected in Tab.1. In the left panel of Tab.1 results from the experiments with changing n (1200 instances in total) are collected, and in the right panel results from the experiments with changing m (700 instances in total) are provided. Heuristics are listed in the order of decreasing number of wins. Only the first 50 positions, out of 70, are shown. Each entry in Tab.1 is a triple: ranking position, name of the heuristic, the number of wins. For example, ILS-A is in the first place in the experiments against changing n and against m, with 880 and 525 wins, respectively. Already the simple comparison of heuristic performance in Tab.1 provides vital insight into algorithm performance. It can be seen in Tab.1 that starting with position 29 the number of wins dramatically collapses and from position 32 the ability to provide the best solutions is 9 times worse (and even more) than in the best method. These bad positions are occupied by the greedy algorithms with *list* and *look-ahead* control structures. The *list* control structure is inefficient because it requires to wait for specific vessels chosen by some sorting rule while ignoring the ready ones. The *look-ahead* control structure suffers similar deficiency: when a future ship arrivals are considered, then they can delay scheduling the currently ready ships giving preference to the future ships elected by the sorting rule order. The best greedy heuristics with *list* or *look-ahead* control structure use FCFS sorting order, thus breaking the spell of waiting for the future ships because FCFS order is the order of the ships arrivals itself. Since *list* or *look-ahead* control structures performed so poorly in this comparison, we will not report on

Table 1: Number of wins in the experiments with changing n and m.

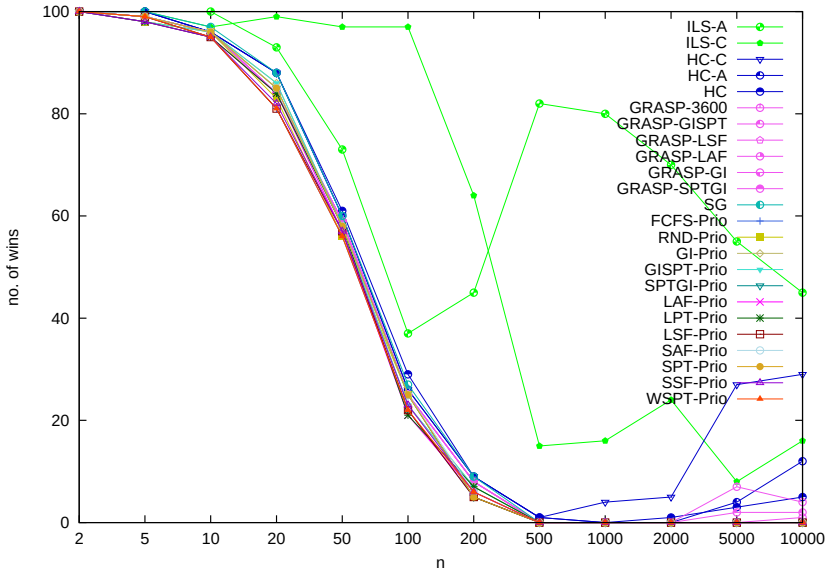
tests vs n, wins in 1200 instances			tests vs m, wins in 700 instances		
1	ILS-A	880	1	ILS-A	525
2	ILS-C	733	2	ILS-C	440
3	HC-C	545	3	HC-C	270
4	HC-A	496	4	HC	268
5	HC	493	5	HC-A	264
6	GRASP-3600	482	6	SG	262
7	SG	481	7	GISPT-Prio	261
8	GRASP-GISPT	475	8	GI-Prio	261
9	GRASP-GI	472	9	SSF-Prio	260
10	GRASP-LSF	471	10	GRASP-SPTGI	259
11	GRASP-LAF	471	11	GRASP-GISPT	259
12	GRASP-SPTGI	471	12	GRASP-LSF	259
13	SAF-Prio	469	13	GRASP-LAF	259
14	SPT-Prio	468	14	GRASP-GI	259
15	GI-Prio	468	15	GRASP-3600	259
16	GISPT-Prio	468	16	RND-Prio	258
17	SPTGI-Prio	468	17	SAF-Prio	257
18	LPT-Prio	463	18	LAF-Prio	256
19	RND-Prio	463	19	SPT-Prio	256
20	SSF-Prio	461	20	LPT-Prio	256
21	LAF-Prio	461	21	WSPT-Prio	256
22	LSF-Prio	459	22	SPTGI-Prio	256
23	WSPT-Prio	459	23	FCFS-Prio	256
24	FCFS-Prio	458	24	FCFS-La2	256
25	FCFS-La2	458	25	FCFS-La5	256
26	FCFS-La5	458	26	FCFS-La10	256
27	FCFS-La10	458	27	FCFS-List	256
28	FCFS-List	458	28	LSF-Prio	254
29	WSPT-La2	370	29	WSPT-La2	93
30	WSPT-La5	260	30	WSPT-La5	54
31	WSPT-La10	165	31	WSPT-La10	36
32	LSF-La2	99	32	LSF-La2	8
33	SSF-La2	93	33	RND-La2	7
34	SPT-La2	92	34	LAF-List	6
35	SPTGI-La2	88	35	LAF-La10	6
36	LPT-La2	87	36	SSF-La5	6
37	GI-La2	86	37	SSF-La2	6
38	GISPT-La2	86	38	LAF-La5	6
39	SAF-La2	82	39	RND-List	6
40	LAF-La2	81	40	LSF-La5	6
41	RND-La2	75	41	LAF-La2	6
42	WSPT-List	60	42	GI-La5	5
43	GI-La5	59	43	GISPT-List	5
44	GI-La10	59	44	GISPT-La10	5
45	GI-List	59	45	WSPT-List	5
46	GISPT-La5	59	46	GISPT-La5	5
47	GISPT-La10	59	47	GISPT-La2	5
48	GISPT-List	59	48	GI-La2	5
49	SSF-La5	56	49	SPTGI-La5	5
50	LPT-List	55	50	LPT-List	5

them in the further discussion. Though the GISPT and GI sorting rules head the list of greedy heuristics, the difference in the number of wins between the heuristic with *priority* control structure is not big. In both experiment series it is roughly 2% in the number of wins. Thus, on average the *priority-based* greedy heuristics perform similarly here. The same conclusion can be drawn for the GRASP methods. The best GRASP method is GRASP-3600 with 741 wins in both experiment series. The Super Greedy method (SG) has similar performance as the GRASP methods, and it is only marginally better than the best greedy methods. Metaheuristics ILS-C, ILS-A dominate in this evaluation by providing 1405, 1173 best solutions, respectively, in both experiment series. In other experiment series the relationships in the number of won instances were similar (not shown here).

In Fig.7a algorithm quality measured by the number of wins is shown against increasing number of ships n. In Fig.7b this relationship is shown for the unique wins. Greedy algorithms have been omitted in Fig.7b because at 1 hour time limit they are dominated by metaheuristics starting from the best greedy solutions, and hence, the greedy methods cannot win uniquely. For each number of ships n, 100 test instances were generated and solved. It can be seen (Fig.7a) that all algorithms find best solutions for almost all small instances ($n = 2, 5, 10$). Thus, small instances are easy to solve, especially considering the number of berths $m \approx 50$ on average. Consequently, it is hard to win uniquely (Fig.7b). RND and FCFS sorting rules can be applied as reference performance indicators, because the first one is oblivious to the relationships existing in the data, while the second just follows the stream of ship arrivals. It can be observed (Fig.7a) that the performance of greedy algorithms is similar to these two reference rules and the Super Greedy algorithm is not an exception here. With increasing n, greedy algorithms lose their ability to provide the best solutions and it ceases at $n \approx 500$, which represents roughly 10 ships per berth. The local search methods ILS, HC, with their variants, start showing their advantage with growing number of the ships. Unfortunately, their disadvantage is computational complexity: Given the fixed runtime limit their ability to exploit the space of possible solutions diminishes with increasing n. Since the ILS-type methods are the most complicated, they lose against a simpler HC-type and the GRASP-type methods for the biggest considered n (Fig.7b). The performance of the GRASP methods, though better than of the greedy algorithms, is rather weak. GRASP-LAF/-LSF/-SPTGI returned no unique best solution. On the other hand, GRASP-3600 method happened to provide 1 best solutions for big instances, which are too hard to deal with for the more complex local search methods (ILS-A/C, HC).

In Fig.8a the performance of the algorithms measured by the number of wins in 100 instances is shown for changing number of berths m. In Fig.8b the number of unique wins is depicted. Methods which returned no unique best solutions, e.g. GRASP-type, are omitted in Fig.8b. It can be observed

a)



b)

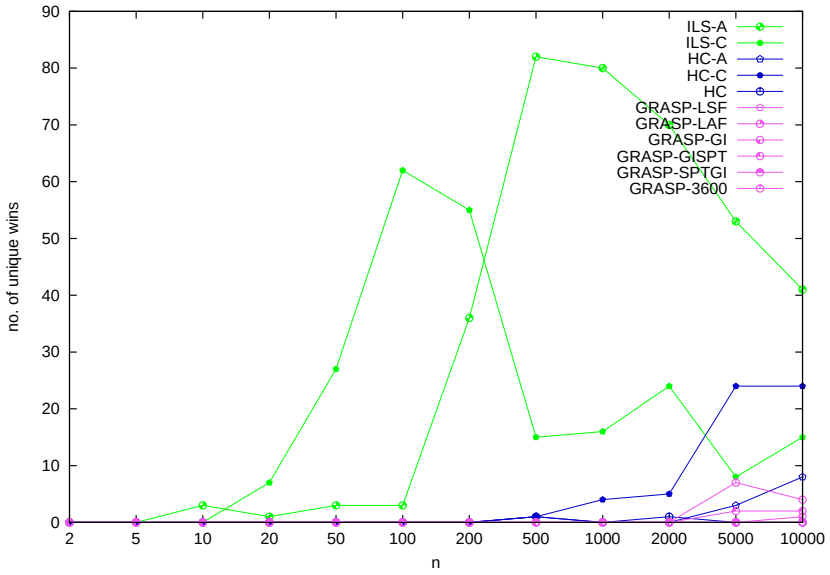
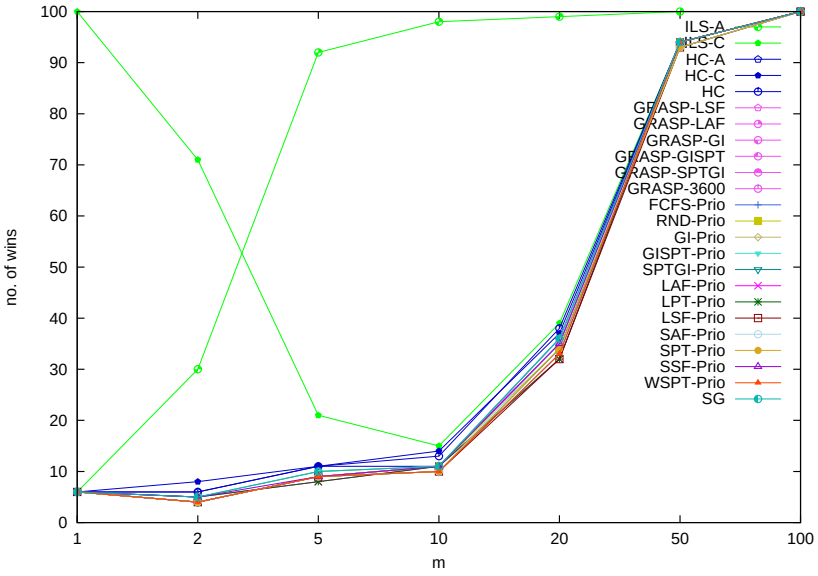


Figure 7: Performance of the algorithms vs n. a) Number of wins, b) number of unique wins.

a)



b)

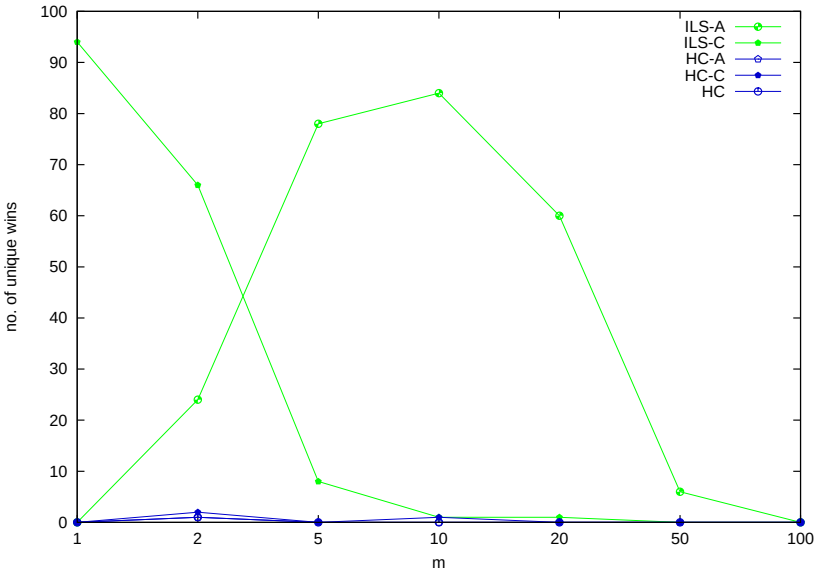


Figure 8: Performance of the algorithms vs m. a) Number of wins, b) number of unique wins.

that with growing number of berths the problem becomes easier to solve and for $m \geq 50$ almost all algorithms provided the best solutions (Fig.8a). This corresponds with roughly 10 ships per berth. On the other end of m range the problem is much harder and most of the greedy methods provide only around 5 best solutions (in 100). ILS-A algorithm does not rebuild the schedules if there is only one berth. Therefore, for $m = 1$ this algorithm performs on par with the greedy algorithms which provide the starting solutions of ILS-A. However, with increasing m ILS-A outperforms other methods. Conversely, ILS-C which is complex and behaves poorly for big m is the best method for $m = 1, 2$. The above phenomena are mirrored in the number of unique wins (Fig.8b). ILS-A is the best method for medium m, while ILS-C is the best algorithm for small m, i.e. when the port is under the biggest load. However, due to its complexity ILS-C loses against simpler methods with increasing m.

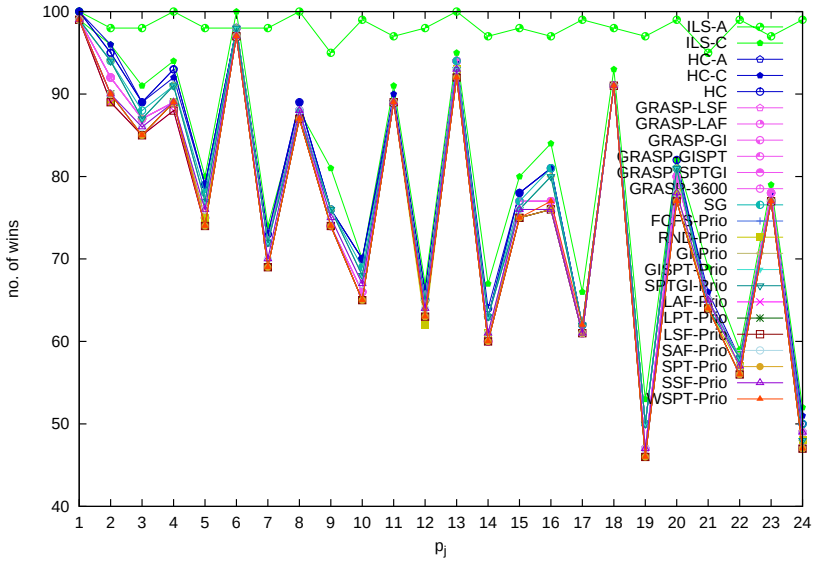
Fig.9 depicts performance of the algorithms against increasing value of p_j. In this series of tests we intended to examine algorithm sensitivity to the growing port load. The swept range of processing times p_j was $[1, 24]$. It follows that each instance had the same processing time for all ships. It can be seen in Fig.9 that for the smallest processing times all methods provide the best solutions because there is hardly any conflict between the ships in a very lightly loaded port. With the growing port load all greedy methods gradually provide worse solutions and there is no much difference between them. ILS-A algorithm is the best here and its advantage grows with the load of the port. This can be attributed to the ability of ILS-A to search the solution space, and its relatively low complexity.

6.3 Quality of the algorithms versus dispersion of instance parameters

The goal of the study on the impact of instance parameter dispersion is to analyze the algorithm susceptibility to the diversity of ship types. Again the algorithms are given 1 hour runtime limit.

In the tests of the impact of ship importance dispersion, w_js were generated from $w_j \sim U[500-\Delta_w, 500+\Delta_w]$, where $\Delta_w \in \{1, 2, 5, 10, 20, 50, 100, 200, 300, 400, 500\}$. For a comparison let us observe that in the previous tests $w_j \sim U[1, 1000]$. Thus, the earlier tests correspond with the greatest Δ_w. The standard deviation of $U[1, 1000]$ is approximately 289 and Δ_w smaller than 289 means restricting w_j to a narrower range than was typical in the earlier tests. The results collected in Fig.10 show the importance of this threshold: When values of w_j are very restricted, no dependency on Δ_w can be observed because other parameters have bigger influence on the constructed solutions. All methods have rather similar performance with some small advantage of ILS-A. Conversely, for larger diversity of ship importance greedy algorithms lose their capability of recreating the best solutions. Con-

a)



b)

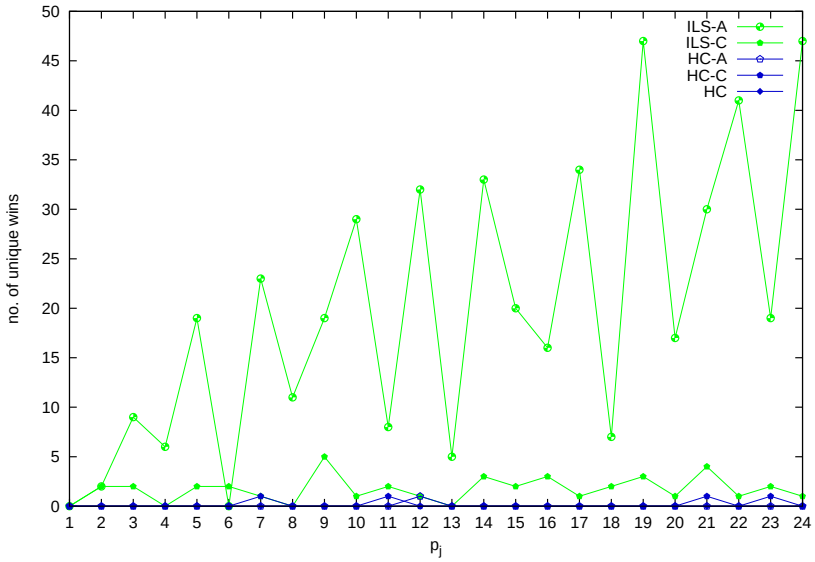
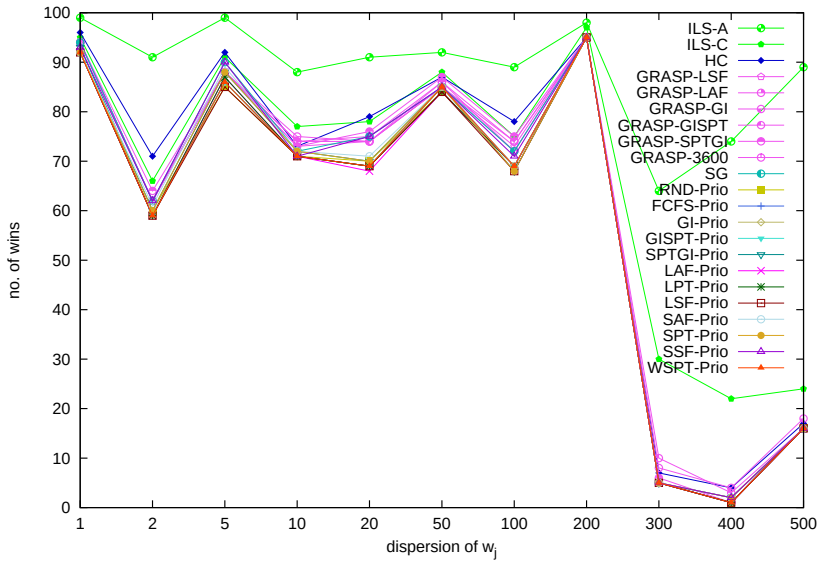


Figure 9: Performance of the algorithms vs p_j. a) Number of wins, b) number of unique wins.

a)



b)

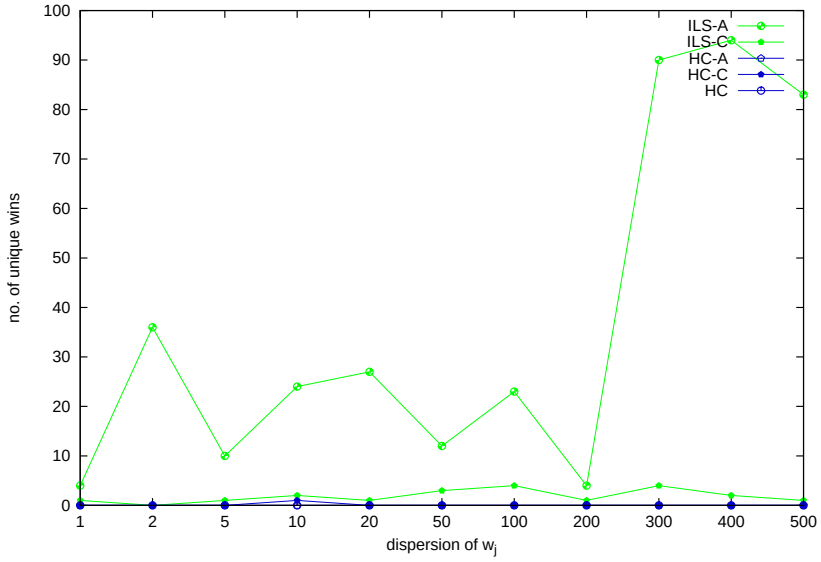


Figure 10: Performance of the algorithms vs dispersion of w_j. a) Number of wins, b) number of unique wins.

Table 2: Heuristic performance for different time limits.

Top 3 heuristics with the highest number of wins						
time limit	1st		2nd		3rd	
100ms	GISPT-Prio:	631	GI-Prio:	597	SAF-Prio:	552
1s	HC-C:	625	HC:	623	HC-A:	585
10s	HC-C:	690	SPTGI-Prio:	656	HC:	616
100s	HC-C:	720	ILS-A:	613	SPTGI-Prio:	594
1000s	ILS-A:	730	HC-C:	697	HC-A:	601
Top 3 heuristics with the highest number of unique wins						
time limit	1st		2nd		3rd	
100ms	GISPT-Prio:	74	SAF-Prio:	49	GI-Prio:	40
1s	SPTGI-Prio:	79	GISPT-Prio:	69	HC-C:	46
10s	SPTGI-Prio:	181	HC-C:	129	HC:	50
100s	HC-C:	176	SPTGI-Prio:	109	ILS-A:	82
1000s	ILS-A:	193	HC-C:	163	ILS-C:	77

sequently, ILS-A/C methods are capable of constructing the best solutions uniquely.

We examined also the impact of ship length dispersions and processing times dispersion (not shown here). In the tests against ship length dispersion, greedy methods were able to win only if the space of possible solutions was very restricted (one, two ship lengths). No significant correlation between the number of wins and dispersion of p_js has been observed. The other results were similar to the ones already shown: Algorithm ILS-A provided the greatest number of best solutions, also exclusively. ILS-C, HC, GRASP methods provided 1-4 unique best solutions (in 100 instances).

6.4 Quality of the solution versus runtime

In the previous experiment series the time cost of constructing solutions has not been considered. However, it is well known that greedy algorithms can return their solutions earlier than more sophisticated heuristics. Conversely, metaheuristics can provide quality results at the cost of longer runtime. Thus, algorithms trade their runtime for solution quality differently. In this section we investigate this dimension of algorithm performance.

Let us first elaborate on the ability of the algorithms to cover a set of instances with the best solutions which is measured by the number of wins and unique wins on the given set of instances. The results of this test are shown in Tab.2. The data were collected in the series of experiments with changing number of ships n (the same as the dataset for Fig.7). For other experiment series the conclusions were similar, so we do not report them here. Algorithms are ranked according to the decreasing number of (unique) wins under runtime limit grades of 0.1s, 1s, 10s, 100s, 1000s. Different time

Table 3: Nondominated heuristics in the $MWFT$ vs runtime comparison (medians).

$n = 100$			$n = 1000$			$n = 10000$		
name	time	quality	name	time	quality	name	time	quality
FCFS-Prio:	9.018ms,	1.055	SPT-Prio:	94.96ms,	1.647	SPT-Prio:	5.417s	1.217
SPTGI-Prio:	9.237ms,	1.040	SPTGI-Prio:	99.51ms,	1.602	SPTGI-Prio:	5.682s	1.192
RND-Prio:	9.338ms,	1.029	GISPT-Prio:	145.9ms,	1.555	GRASP-SPTGI:	549.9s	1.083
SSF-Prio:	10.28ms,	1.026	GI-Prio:	146.2ms,	1.551	HC-A:	753.3s	1.033
SG:	435.1ms,	1.0209	SG:	7.051s,	1.463	HC:	1158s	1.011
HC-A:	435.1ms,	1.0209	HC-A:	7.162s,	1.354	HC-C:	1403s	1.007
HC-C:	467.5ms,	1.0207	HC:	8.232s,	1.298	ILS-A	3351s	1.0002
HC:	475.4ms,	1.0195	HC-C:	8.355s,	1.289			
ILS-C:	2958s,	1	ILS-A:	1731s,	1			

grades represent various levels of acceptable runtime in solving long-time horizon BAP. Let us observe that imposing runtime limits may eliminate some methods from the ranking, because some algorithms may be unable to obtain any solution on certain instances within the time limit. For example, methods starting from the best greedy solution (HC-type, ILS-type) may not to even start their search if problem sizes are big or time limits are tight. In Tab.2 best three algorithms in the sense of (unique) wins are provided with their number of (unique) wins in 1200 instances. It can be seen that for low time limits the greedy algorithms dominate. With the increasing runtime limit, metaheuristics come forward and HC dominates because it is faster than the more sophisticated local search method (ILS-A/C). Super Greedy and GRASP methods do not appear at the top of the ranking because for low runtime limits they are too slow, while their starting solutions are derived by the greedy algorithms, and for longer runtimes local search methods (HC, ILS-A/C) return better solutions.

The criteria applied earlier, namely the number of (unique) wins, represent algorithm ability to cover diverse sets of instances. But still, the instances are not equal and these two criteria do not reveal the distance from the best solution in terms of the $MWFT$ goal function. Therefore, $MWFT$ has been applied as the quality indicator in another examination of the time-quality trade-off offered by the algorithms. The outcome of the evaluation is collected in Fig.11 and in Tab.3. In Fig.11 quality of the solutions vs runtime is shown. The time until delivering the best solution constructed is depicted in Fig.11 (not the total runtime which is 1 hour in the case of metaheuristics). The data were obtained on a set of 1200 instances in the tests against changing n. For each heuristic a median and whiskers extending from the first quartile to the third quartile in quality (vertically) and in time (horizontally) are shown. The ranges are normalized to 1 hour in time and to the best solution obtained by any algorithm in quality. This means that value 1 on the vertical axis represents the best solution found. Let us

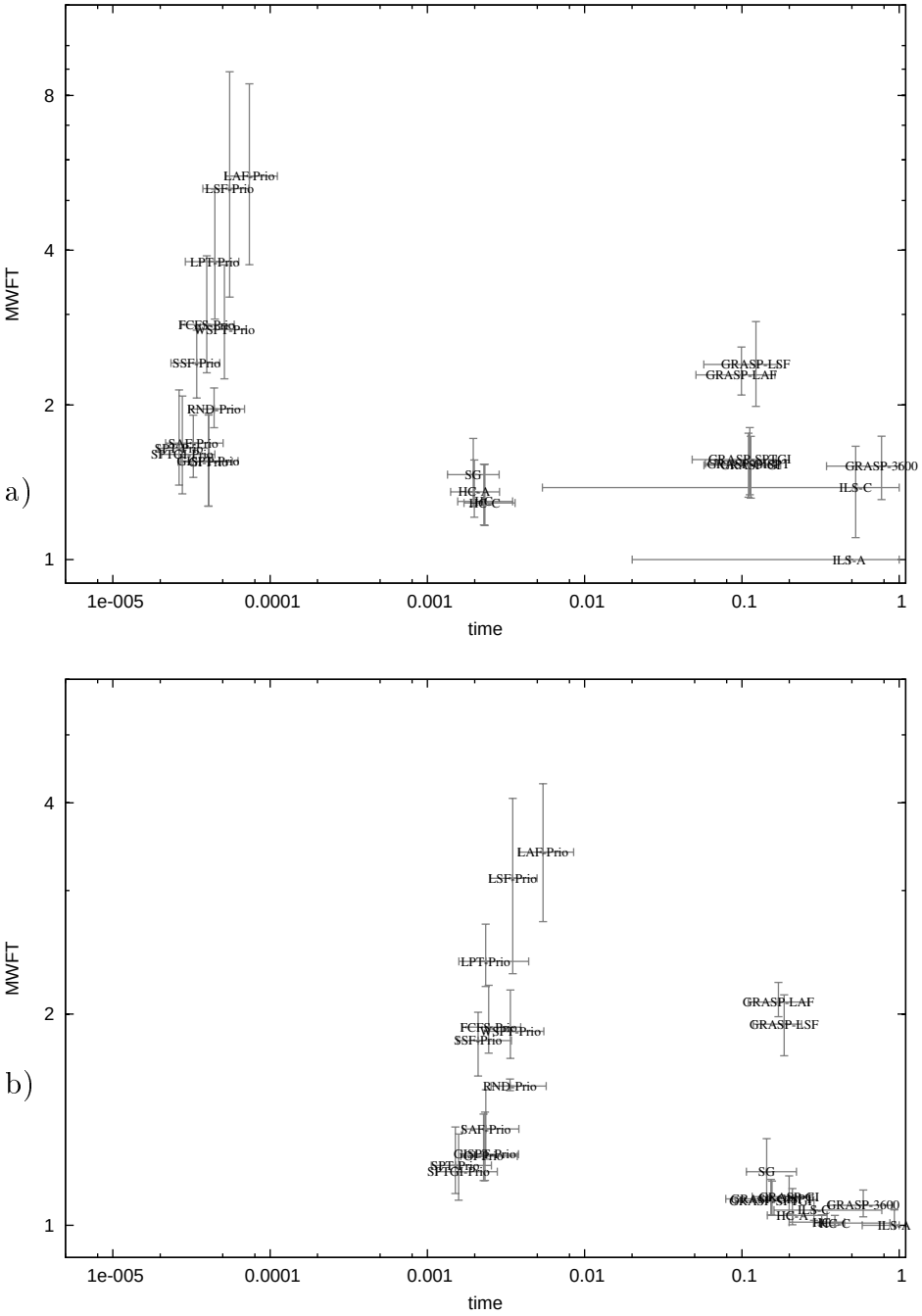


Figure 11: Quality ($MWFT$) vs runtime. a) $n = 1000$, b) $n = 10000$.

observe that the lower-left corner in Fig.11 is the direction of the desired algorithm optimization in this bicriterion setting. For clarity of presentation, the set of nondominated algorithms is shown also in Tab.3. Fig.11a presents results for $n = 1000$, and Fig.11b for $n = 10000$, which gives insight into the evolution of the quality-vs-time trade-off with changes of the instance sizes. With growing n the times of returning the best constructed solutions grow in all methods. It can be observed (Fig.11) that many greedy algorithms are dominated, even by the greedy algorithm with the RND sorting rule. Their randomized versions: GRASP-LSF/-LAF/-3600 are not much better in solution quality, but still they take more time to run. These algorithms are impractical. Moreover, the relationship in performance between weak and good sorting rules in the greedy algorithms is not changed by embedding the rules in the GRASP methods. Only the algorithms based on SPT, SPTGI, GI, GISPT rules can be found in the set of nondominated algorithms. Taking into account the dispersion of time and quality indicators, the overall performance of these four greedy algorithms can be regarded similar. Other greedy algorithms are competitive only for $n = 100$ (Tab.3, FCFS, SSF), which is hard to consider a long time horizon instance size. A more profitable approach is bundling the greedy algorithms in Super Greedy (SG) algorithm. But still, SG appears non-dominated only for medium runtime budgets and small-medium problem sizes. The GRASP-type methods are almost always dominated, with the exception of GRASP-SPTGI coming forward to quality-time Pareto front at $n = 10000$ (see Tab.3). ILS-A is competitive for medium-large size instances and long runtime (Fig.11a, b).

Tab.2 and Tab.3 are a good summary of this series of experiments and a recommendation for selection of the algorithms solving big BAPs: For really big BAP instances or short runtime budget priority greedy algorithms based on SPT, SPTGI, GISPT, GI sorting rules are the best choice. If the runtime limits are more lenient, local search algorithms can be applied: HC-C, ILS-A (in the order of increasing runtime and improving solution quality).

7 Conclusions

In this work we considered the algorithms solving large size BAP problems with tight runtime budget limitations. It has been discovered that greedy algorithms with *list, look-ahead* control structures are impractical because they give preference to the order of some sorting rule while ignoring the ships ready to be served. Choosing the best solution built by a set of greedy algorithms (Super Greedy, SG) or applying their randomized versions (GRASP) is rarely practical. However, SG can be improved by restricting the set of component greedy algorithms, thus creating a faster algorithm without sacrificing much of solution quality. Metaheuristics, such as the local search-based, are capable of constructing good solutions only if given enough time.

The hill climber-type is the fastest among them and has the widest applicability under runtime restrictions. If the runtime limit is getting tight or problem size is large, one has to recourse to simple methods like hill climber, and finally greedy heuristics. Inevitably, this results in constructing inferior quality solutions. Still, it can be considered acceptable if the data about future streams of ships are uncertain while many alternative scenarios must be quickly verified.

We believe that the study presented in this paper may be interesting also from the methodological point of view because a non-standard approach to the evaluation of heuristics has been proposed. Different quality criteria have been applied ($MWFT$, number of wins). The performance of the heuristics has been evaluated in the time-vs-quality trade-off. We depart from the idea of "one best" algorithm to solve any instance of the problem, but rely instead on a toolbox of alternative algorithms applicable in different settings. An extension of this work may progress in at least two directions. On the one hand, the selected algorithms can be exploited in long-term port simulation and estimation of the port throughput. On the other hand, the process of selecting the set of usable heuristics should be automated such that the human expert is taken out of the decision process.

Acknowledgements

All computational experiments have been conducted in Poznań Supercomputing and Networking Center. This research has been partially supported by a grant of French National Research Ministry and PHC Polonium project for years 2017-2018.

References

[1] Christian Bierwirth and Frank Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615 – 627, 2010.

[2] Christian Bierwirth and Frank Meisel. A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 244(3):675 – 689, 2015.

[3] Iwo Błądek, Maciej Drozdowski, Frédéric Guinand, and Xavier Schepler. On contiguous and non-contiguous parallel task scheduling. *Journal of Scheduling*, 18(5):487–495, 2015.

[4] Katja Buhrkal, Sara Zuglian, Stefan Ropke, Jesper Larsen, and Richard Lusby. Models for the discrete berth allocation problem: A compu-

tational comparison. *Transportation Research Part E: Logistics and Transportation Review*, 47(4):461 – 473, 2011.

[5] Edward Coffman and Michael Garey. Proof of the 4/3 conjecture for preemptive vs. nonpreemptive two-processor scheduling. In *STOC '91 Proceedings of the 23rd annual ACM symposium on Theory of computing*, pages 241–248, 1991.

[6] Jean-François Cordeau, Gilbert Laporte, Pasquale Legato, and Luigi Moccia. Models and tabu search heuristics for the berth-allocation problem. *Transportation Science*, 39(4):526–538, 2005.

[7] Giovanni Giallombardo, Luigi Moccia, Matteo Salani, and Ilaria Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232 – 245, 2010.

[8] Pierre Hansen and Ceyda Oğuz. A note on formulations of the static and dynamic berth allocation problems. Technical Report G-2003-30, GERAD, 2003.

[9] Maarten Hendriks, Marco Laumanns, Erjen Lefeber, and JanTijmen Udding. Robust cyclic berth planning of container vessels. *OR Spectrum*, 32(3):501–517, 2010.

[10] M.P.M. Hendriks, D. Armbruster, M. Laumanns, E. Lefeber, and J.T. Udding. Strategic allocation of cyclically calling vessels for multi-terminal container operators. *Flexible Services and Manufacturing Journal*, 24(3):248–273, 2012.

[11] Akio Imai, Yukiko Yamakawa, and Kuancheng Huang. The strategic berth template problem. *Transportation Research Part E: Logistics and Transportation Review*, 72:77 – 100, 2014.

[12] Eduardo Lalla-Ruiz, Beleń Meliań-Batista, and J.Marcos Moreno-Vega. Artificial intelligence hybrid heuristic based on tabu search for the dynamic berth allocation problem. *Engineering Applications of Artificial Intelligence*, 25(6):1132–1141, 2012.

[13] Pasquale Legato, Rina Mary Mazza, and Daniel Gullì. Integrating tactical and operational berth allocation decisions via simulation-optimization. *Computers & Industrial Engineering*, 78:84 – 94, 2014.

[14] Rajeeva Moorthy and Chung-Piaw Teo. Berth management in container terminal: the template design problem. *OR Spectrum*, 28(4):495–518, 2006.

[15] Xavier Schepler. *Solutions globales d'optimisation robuste pour la gestion dynamique de terminaux à conteneurs*. PhD thesis, Université du Havre, 2015.

[16] Xavier Schepler, Stefan Balev, Sophie Michel, and Eric Sanlaville. Global planning in a multi-terminal and multi-modal maritime container port. *Transportation Research Part E, under revision*, 2016.

[17] Pierre Thorez and Oliver Joly. Port competition in the northern range from le havre to hamburg. *PROMET-Traffic&Transportation*, 18(2): 77–82, 2012.

[18] Jose Tongzon and Wu Heng. Port privatization, efficiency and competitiveness: Some empirical evidence from container ports (terminals). *Transportation Research Part A: Policy and Practice*, 39(5):405 – 424, 2005.

[19] Bert Vernimmen, Wout Dullaert, and Steve Engelen. Schedule unreliability in liner shipping: origins and consequences for the hinterland supply chain. *Maritime Economics & Logistics*, 9(3):193–213, 2007.

[20] Bart W. Wiegmans, Anthony Van Der Hoest, and Theo E. Notteboom. Port and terminal selection by deep-sea container operators. *Maritime Policy & Management*, 35(6):517–534, 2008.

[21] Lu Zhen. Tactical berth allocation under uncertainty. *European Journal of Operational Research*, 247(3):928 – 944, 2015.

[22] Lu Zhen, Ek Peng Chew, and Loo Hay Lee. An integrated model for berth template and yard template planning in transshipment hubs. *Transportation Science*, 45(4):483–504, 2011.