VRP-Inspired Techniques for Discrete Dynamic Berth Allocation and Scheduling

Konstantinos Karathanasis
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Spyros Kontogiannis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press Diophantus, Patras, Greece

Asterios Pegos
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Vasileios Sofianos
Department of Computer Engineering and Informatics, University of Patras, Greece
PIKEI New Technologies, Patras, Greece

Christos Zaroliagis
Department of Computer Engineering and Informatics, University of Patras, Greece
Computer Technology Institute and Press Diophantus, Patras, Greece

Abstract

The *Berth Allocation and Scheduling Problem* (BASP) is a critical optimization challenge in maritime logistics, aiming to assign arriving vessels to berths efficiently, while adhering to practical constraints. Exploiting the connection of BASP with the *Heterogeneous Vehicle Routing Problem with Time Windows* (HVRPTW), we propose a mixed integer linear programming (MILP) formulation for a variant of BASP which is of utmost importance in real-world scenarios: the *Dynamic Discrete Berth Allocation and Scheduling Problem with Time Windows* (DDBASPTW). Consequently, inspired by the wealth of constructive and improvement heuristics for VRP, we design, implement and experimentally evaluate three constructive heuristics, *Nearest Neighbour* (NN), *Insertion* (INS), a quick-and-dirty variant of Insertion (qd-INS), as well as two improvement heuristics, *Swap* and *Reinsert*, taking into consideration both the online and the offline scenario with respect to vessel arrivals. Finally, we propose, implement and experimentally evaluate, custom-tailored variants for DDBASPTW of a single-solution metaheuristic, the *Adaptive Large Neighborhood Search* (ALNS), and of two population-based metaheuristics, the *Genetic Algorithm* (GA) and the *Cuckoo Search Algorithm* (CSA), which are aimed to solve the offline version of the problem. An extensive experimental evaluation compares these techniques against a generic state-of-the-art MILP solver. Results demonstrate that certain variants of INS not only are extremely fast and deliver competitive solutions, achieving a practical trade-off between execution times and quality of solutions. The improvement heuristics further refine the initial solutions, especially for weaker constructive approaches, offering a lightweight yet effective enhancement mechanism. The metaheuristics consistently yield high-quality solutions with significantly lower computational times compared to the exact MILP solver, making them well-suited for use in real-time or large-scale operational environments.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases Berth Allocation and Scheduling, Heuristics, Metaheuristics, Mixed Integer Linear Programming

Digital Object Identifier 10.4230/OASIcs.ATMOS.2025.6

Funding This work is partially supported by the EU I3 Instrument under GA No 101115116 (project AMBITIOUS) and by the University of Patras under GA No 83770 (programme "MEDICUS").

1 Introduction

In recent decades, the volume of goods that is transported globally has shown consistent growth [10], reflecting similar trends in maritime trade. Approximately 12 billion tons of traded goods were transported by sea in 2022, accounting for roughly 49% of total trade, nearly double the proportion transported by air [11]. The ongoing growth of maritime trade introduces significant challenges for marine container terminal operators. These challenges must be addressed using appropriate strategies to meet market demands. Therefore, more efficient management of resources and berthing slots is required, so as to improve service quality, accommodate a greater number of vessels, and reduce berthing costs. Employing optimized techniques for berth allocation and scheduling is essential to achieving these goals.

The *Berth Allocation and Scheduling Problem* (BASP) aims to provide a service schedule for each arriving vessel, offering the optimal berthing position and mooring time, while avoiding collisions and adhering to all practical constraints. The variants of BASP are classified in three categories, depending on the spatial characteristics of the wharves: *discrete*, continuous, or hybrid [7] (cf. Figure 1). In the discrete version, a wharf is divided into distinct berthing segments, each capable of accommodating a single vessel at a time, subject to spatial constraints. In contrast, in the continuous version of the problem, a vessel can moor anywhere along the length of the wharf, with berthing positions assigned according to the specific requirements of each vessel rather than fixed positions. Finally, in the hybrid version, the wharf is divided into distinct berthing positions, but a vessel may occupy multiple segments or share a berthing segment with another vessel simultaneously. The variants of

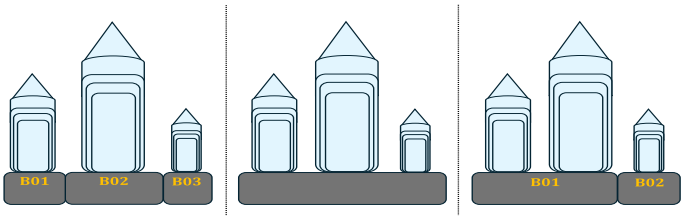


Figure 1 Illustration of berthing layouts: Discrete (left), Continuous (middle), and Hybrid (right).

BASP are also classified depending on the expected arrival times of vessels, as either *static*, *dynamic*, or *uncertain*. In static BASP, it is assumed that all vessels already arrived at the port and are awaiting service. In dynamic BASP, the vessels are assumed to arrive at the port at different times, but their arrival times may be either unknown (online scenario) or a priori known (offline scenario) to the scheduler. In the uncertain case, vessel arrival times are subject to uncertainty due to external factors. By combining the previously mentioned categorizations, specific problem variants can be derived, which are encoded in this paper as XYBASP for $X \in \{$D(iscrete), C(ontinuous), H(ybrid)$\}$ and $Y \in \{$D(ynamic), S(tatic), U(ncertain)$\}$. E.g., DDBASP is the discrete dynamic variant, CSBASP is the continuous static variant, etc. Interestingly, DDBASP shares notable similarities with the *Vehicle Routing Problem* (VRP) [17], a foundational optimization problem in logistics, focusing on determining efficient routes for a fleet of vehicles to service a set of customers. VRP aims to minimize operational costs, e.g., travel distance or time, while respecting several spatiotemporal constraints for the involved stakeholders (workers and customers). Challenges of real-world logistics have led to numerous variations of VRP, including the *Vehicle Routing Problem with Time Windows* (VRPTW) which incorporates temporal requirements for customer service, and the *Heterogeneous Vehicle Routing Problem* (HVRP) which accounts

for fleets of vehicles with varying capacities, costs, and characteristics. The combination of these two variations results in the *Heterogeneous Vehicle Routing Problem with Time Windows* (HVRPTW), a highly realistic and challenging variant, which tries to balance vehicle heterogeneity with temporal constraints, and is especially relevant in applications like urban logistics and e-commerce.

Related Work. BASP is a critical optimization problem in port operations due to its significant impact on efficiency and costs. Since this is a computationally challenging (NP-hard) problem [8], exact methods are rather inapplicable in real-world scenarios because of the increasing complexity of modern BASP variants. As a consequence, approximate and heuristic methods are often preferred in practice. The majority of research on DDBASP has mainly focused on exact solvers and generic metaheuristics. To our knowledge, there is lack of extensive experimental evaluations of various heuristic techniques which are custom-tailored for DDBASP, as is for example the case for other well-known fundamental optimization problems (like TSP and VRP).

Three mathematical models for DDBASP were considered and experimentally evaluated in [3], one of which being based on a VRPTW formulation of the problem. However, this work has several limitations: (i) It does not account for vessels and berths of different sizes, whereas many real-world instances involve vessels of varying types and berths of different spatial characteristics and operational capabilities. (ii) It does not allow for denial of service for some vessels, rendering many real-world instances infeasible, although daily practice is actually based on best-effort approaches. (iii) It does not distinguish between soft deadlines (e.g., requested departure times by the vessels) and hard time windows (e.g., commitment of the port authority to deliver service no later than a maximum delay beyond the requested departure time), which is also typical practice in daily port management. (iv) Only generic exact solvers were considered and experimentally evaluated on the proposed DDBASP models.

A self-adaptive evolutionary algorithm was introduced in [5] to solve a *Mixed-Integer Linear Programming* (MILP) formulation for DDBASP, aimed at minimizing a weighted sum of waiting times, handling times and late departures, which is then compared against a generic exact solver. Moreover, a mathematical formulation for DDBASP was also introduced in [13] and a *Bee Colony Optimization*-based metaheuristic was deployed for its solution, which was experimentally evaluated against an exact solver. Nevertheless, in both these approaches no denial of service to vessels is allowed, no soft and hard deadlines were considered, and there is no comparison with other heuristic and metaheuristic approaches which are custom-tailored for DDBASP.

A MILP model for the continuous variant, CDBASP, was introduced in [1, 2], aiming to minimize the total turnaround cost. Moreover, the *Cuckoo Search Algorithm* (CSA) was proposed and experimentally evaluated on relatively small instances. Nevertheless only high-level information is provided in [1, 2] for creating the initial population and for the reproduction process which, if applied verbatim, would almost certainly lead to infeasible solutions, especially for the discrete variants of BASP that is studied in our work.

Recent contributions further illustrate that berth allocation remains an active and evolving research topic. For example, the berth allocation and quay crane assignment problem with crane travel and setup times is studied in [4], proposing MILP formulations and hybrid exact/metaheuristic approaches for solving them. More recently, berth allocation and time-varying quay crane scheduling during emergencies is investigated in [15], introducing a bi-objective MILP formulation and an improved particle swarm optimization algorithm to solve it. These works highlight the increasing interest of the community in developing

sophisticated optimization models and algorithms for BASP. However, their focus is on integrated berth-crane assignment and energy-related objectives, which is different from the problem that is considered in the present work.

Our Contribution. The focus of our study is to propose and experimentally evaluate heuristic and metaheuristic techniques which are custom-tailored for DDBASP under real-world considerations, such as potential denial of service, heterogeneity of both vessels and berths, and consideration of soft and hard deadlines. Towards this direction, we formulate *DDBASP with Time Windows* (DDBASPTW) as a generalization of the HVRPTW formulation proposed in [3], enabling the deployment and adaptation of well-established heuristic techniques from vehicle routing to berth allocation. Each vessel should be assigned for service to an eligible berth for a unique (continuous) time interval, causing a vessel-dependent servicing cost which accounts for waiting times, handling times and late departures from the berths. Otherwise, vessels that are declined service incur service-rejection costs. Exploiting our formulation, we then adjust two well-known constructive heuristics for VRP instances to address the challenges of DDBASPTW, namely, *Nearest Neighbour* (NN) and *Insertion* (INS). We also introduce the *quick-and-dirty Insertion* (qd-INS) heuristic, which provides competitive but much faster solutions than INS. For each heuristic we implement two versions: the *offline* version which utilizes the complete knowledge of vessel arrivals to determine a solution, and the *online* version where the vessels are assigned to berths, one after the other, immediately when they are revealed to the system in a predetermined order (in our case, increasing arrival times). Additionally, we adjust to DDBASPTW two improvement heuristics for VRP, namely, *Reinsert* and *Swap*, which are used to explore the neighborhood of incumbent solutions of the previously mentioned constructive heuristics for local improvements. Finally, we deploy for DDBASPTW the single-solution metaheuristic *Adaptive Large Neighborhood Search* (ALNS), as well as two population-based metaheuristics, the *Cuckoo Search Algorithm* (CSA) and the *Genetic Algorithm* (GA), aiming to provide solutions of near-optimal quality within reasonable time.

In all our techniques, an elementary operation that is repeatedly executed and thus constitutes a crucial bottleneck for their efficiency, is the *detection of potential conflicts* of a candidate servicing interval for a particular vessel (e.g., a newly arrived vessel that must be assigned to some berth, or an already assigned vessel that is to be relocated in hope for an improved solution) with an existing berth schedule. We have chosen to represent berth schedules as *interval trees*, so as to efficiently conduct these conflict detections.

We also conduct a thorough experimental evaluation comparing all our techniques w.r.t. their computational requirements and the quality of the proposed solutions, with the optimal solutions provided by a generic branch-and-cut MILP solver of Gurobi[1]. For the needs of our experiments we introduce a synthetic data generation process, which in our opinion is of independent interest, that creates large synthetic data sets whose characteristics resemble those of small real-world benchmark data sets that we have at our disposal. The experimental results demonstrate that our constructive heuristics for DDBASPTW exhibit extremely fast execution times with acceptable solution quality, and often achieving high-quality results. Among them, the variants of INS outperform those of NN. The experimental evaluation also demonstrates the effectiveness of our metaheuristic approaches, with ALNS consistently delivering high-quality solutions for this particularly challenging problem. Compared to the constructive heuristics, all the metaheuristic techniques are definitely slower but provide

[1] `https://www.gurobi.com`

significantly better solutions. In several instances, ALNS even finds the optimal solution, and achieves gaps no greater than 29% across all our data sets. This is a remarkable performance, given that even the exact solver of Gurobi reports optimality gaps ranging up to 8% in some data sets. In addition to its strong solution quality, ALNS significantly outperforms the MILP solver in terms of execution times, achieving speedups ranging from 353 up to 83076 among our experimental data sets. Finally, our improvement heuristics for DDBASPTW consistently enhance incumbent solutions whenever there is indeed room for such an improvement, achieving noticeable quality gains with minimal computational overhead, thereby complementing both constructive heuristics and metaheuristics within our solution framework.

2 Problem Description & Mathematical Formulation

In this section, we introduce a mathematical model formulating a DDBASPTW instance as a generalization of an HVRPTW instance. HVRPTW is modelled on a directed graph $G = (V, E)$, where the set of nodes $V = S \cup \{o, d\}$ includes a set S of nodes representing customer-service points, assigned to an eligible vehicle from a fleet B, as well as two special nodes o and d representing an origin-depot and a destination-depot for all the vehicles of B; the set of (directed) edges is $E = (o \times S) \cup (S \times d) \cup (S \times S) \cup (o, d)$. The task of each vehicle is then represented as an (o, d)-*subtour*, i.e., a sequence of nodes from V starting at o and ending at d, with intermediate nodes corresponding to customer-service points to be served in a particular order.

Table 1 Notation.

Category	Symbol	Description
Sets, Indices and Special Nodes	$i, k \in S$	Indices that indicate (as subscripts) vessels from S.
	$j \in B$	Index that indicates (as superscript) berths from B.
	o, d	Origin-depot and destination-depot for a VRP instance, corresponding to initial-state and final-state of each berth.
Variables	$\boldsymbol{x_{i,k}^{j}}$	Indicator variable declaring that $i, k \in S$ are both assigned to berth $j \in B$ for their service, with k being served right after the completion of i's service.
	$\boldsymbol{t_i^j}$	Mooring time of $i \in S$ at berth $j \in B$ to start being served.
	$\boldsymbol{h_i}$	Handling time for $i \in S$ at the berth to which it has been assigned.
	$\boldsymbol{t_i^{ld}}$	Late departure time for $i \in S$ from the berth to which it was served.
Parameters	T_i^{ea}	Estimated Time of Arrival for vessel $i \in S$.
	T_i^{rd}	Requested Time of Departure for $i \in S$.
	H_i^j	Handling time for $i \in S$ at berth $j \in B$.
	N^j	Number of cranes operating at berth $j \in B$.
	T	Time of a single crane to load/discharge a unit quantity.
	Q_i	Cargo Quantity of vessel $i \in S$.
	P_i	Penalty associated with $i \in S$.
	L_i, D_i	Length and draft of vessel $i \in S$, respectively.
	L^j, D^j	Length and draft of berth $j \in B$, respectively.
	C_i^w, C_i^h, C_i^{ld}	Per time-unit costs for waiting, handling and late departure, respectively.
	T^{dp}	Delay Policy (time units)
	M	Sufficiently large positive integer

We adapt such a HVRPTW instance to our DDBASPTW context as follows: Each vehicle is perceived as a berth $j \in B$, while the customer-service points correspond to vessels waiting to be moored and served by a berth. Each vessel $i \in S$ has an estimated time of arrival T_i^{ea} and a requested time of departure T_i^{rd}. The terminal enforces a delay policy T^{dp}, representing the maximum delay beyond a vessel's requested time of departure. This results in a (hard) time window $[T_i^{ea}, T_i^{rd} + T^{dp}]$ within which vessel i must be moored to a berth and complete its service, otherwise it is rejected. Each vessel $i \in S$ carries a cargo quantity Q_i and a desirable number of cranes which could ideally be used for its service. Furthermore, vessel i is associated with a penalty P_i, representing the total loss incurred by the terminal operator if it is eventually decided to deny servicing it. It is also associated with three servicing cost parameters, C_i^w, C_i^h, C_i^{ld}, representing the per-unit time costs for waiting to be moored, for being handled at some berth, and for its departure being delayed beyond T_i^{rd}, respectively.

Regarding the infrastructure, the wharf is assumed to be partitioned into a set B of heterogeneous berths, each berth $j \in B$ differing in its length L^j, draft D^j and number of operational cranes N^j. All cranes are assumed to operate at the same rate, i.e., they require an equal amount of time T to serve (i.e., load or unload) one unit of cargo.

The assignment of a subset of vessels $S_j \subseteq S$ to a particular berth $j \in B$ in a specific servicing order is represented by its *subtour* P_j, i.e., a sequence of nodes from V (with no repetitions), starting from depot o, then "visiting" the nodes in S_j (representing the vessels) in a particular order, and finally ending at depot d. Note that P_j represents just a simple (o, d)-path in G, and every edge $(i, k) \in E$, if included in P_j, declares the immediate precedence of vessel i (or the origin) over vessel k (or the destination), in j's servicing order. Each vessel $i \in S$ is also characterized by its length L_i and draft D_i, which must be compatible with the corresponding attributes of the berth to which it is assigned. Assigning vessel i to a berth $j \in B$ may lead to waiting times, handling times and departure delays. The waiting time of i refers to the delay between its arrival time and the start of its mooring at j. The handling time of i is determined by the specific vessel-berth combination, reflecting the duration required by j to serve (e.g., load or unload) i, based on i's cargo and on j's operational parameters. A departure delay for i occurs when its service by j finishes after its requested departure time. The objective is to minimize the total cost incurred from vessel waiting times, handling times and departure delays, while abiding with all the spatiotemporal constraints, as well as the penalties for vessels that are eventually not assigned to any berth. The feasible solutions of the model consist of a collection of non-overlapping subtours (i.e., internally vertex-disjoint (o, d)-paths in G) for all berths. To formalize this problem, the following assumptions are made: (i) the total number of vessels arriving within the planning horizon is known in advance; (ii) each vessel requires a single, continuous time interval for being handled; (iii) all berths are empty at the start of the planning horizon.

Our MILP model for DDBASPTW (cf. Figure 2 for the MILP, and Table 1 for the used notation) considers two types of variables. The first type are (boolean) decision variables $\boldsymbol{x_{i,k}^j}$, defined for triples $(i, k, j) \in S \times S \times B$, indicating whether vessel k immediately follows vessel i in berth j's subtour. The second type are (continuous) temporal variables defined for pairs $(i, j) \in S \times B$, representing, respectively, the mooring time $\boldsymbol{t_i^j}$ of vessel i at berth j (being equal to T_i^{ea} if i is not assigned to j), the handling time $\boldsymbol{h_i}$ (i.e., the duration of its servicing) of i and the delay $\boldsymbol{t_i^{ld}}$ (i.e., beyond T_i^{rd}) in i's departure time right after the completion of its service, by the berth to which it was moored. The objective function (1) aims to minimize the aggregate costs for all vessels' waiting times, handling times and departure delays, or penalties when denying service. Constraint (2) requires that each vessel

$$\text{minimize} \quad \sum_{i\in S}\sum_{j\in B} C_i^w \cdot (\boldsymbol{t_i^j} - T_i^{ea}) + \sum_{i\in S}(C_i^{ld} \cdot \boldsymbol{t_i^{ld}} + C_i^h \cdot \boldsymbol{h_i}) + \sum_{i\in S} P_i \cdot \Big(1 - \sum_{j\in B} \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j}\Big) \tag{1}$$

s.t. :

$$\sum_{j\in B} \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq 1, \qquad \forall i \in S \tag{2}$$

$$\sum_{k\in S\cup\{d\}} \boldsymbol{x_{o,k}^j} = 1, \qquad \forall j \in B \tag{3}$$

$$\sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} = \sum_{k\in S\cup\{o\},\ i\neq k} \boldsymbol{x_{k,i}^j}, \qquad \forall i \in S,\ \forall j \in B \tag{4}$$

$$L_i \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq L^j, \qquad \forall i \in S,\ \forall j \in B \tag{5}$$

$$D_i \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j} \leq D^j, \qquad \forall i \in S,\ \forall j \in B \tag{6}$$

$$\boldsymbol{h_i} \geq H_i^j \cdot \sum_{k\in S\cup\{d\},\ i\neq k} \boldsymbol{x_{i,k}^j}, \qquad \forall i \in S,\ \forall j \in B \tag{7}$$

$$\boldsymbol{t_i^j} \geq T_i^{ea}, \qquad \forall i \in S,\ \forall j \in B \tag{8}$$

$$\boldsymbol{t_i^j} + \boldsymbol{h_i} \leq T_i^{rd} + T^{dp}, \qquad \forall i \in S, \forall j \in B \tag{9}$$

$$\boldsymbol{t_i^j} + \boldsymbol{h_i} - \boldsymbol{t_k^j} \leq (1 - \boldsymbol{x_{i,k}^j}) \cdot M, \qquad \forall i,k \in S,\ \forall j \in B \tag{10}$$

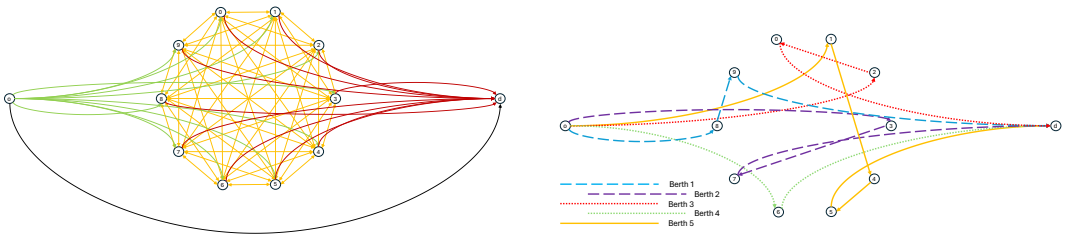
$$\boldsymbol{t_i^{ld}} = \max\{\boldsymbol{t_i^j} + \boldsymbol{h_i} - T_i^{rd}, 0\}, \qquad \forall i \in S, \forall j \in B \tag{11}$$

Figure 2 MILP Formulation for DDBASPTW. Parameters in **boldfaced small letters** (e.g., $\boldsymbol{x_{i,k}^j}$, $\boldsymbol{t_i^j}$, or $\boldsymbol{h_i}$) indicate (decision, or temporal) variables to the MILP. Parameters in capital letters (e.g., T_i^{ea}, T^{dp}, or H_i^j) indicate constants, which are provided as input.

is assigned to at most one berth, thereby preventing double bookings. Observe that no two consecutive vessels in a subtour can be identical, which is reflected by having $i \neq k$. Constraint (3) ensures that each berth's shift begins from the origin-depot o and finishes at the destination-depot d. Observe that each berth will definitely start and complete its own shift, even if the corresponding subtour goes directly from the origin-depot o to the destination-depot d, resulting in no vessels being serviced by it. Constraint (4) guarantees the continuity of the servicing process by maintaining flow conservation at each vessel's servicing point: Each berth enters and exits a vessel's servicing point exactly the same number of times. The *spatial* constraints for length-eligibility (5) and draft-eligibility (6), extend beyond the traditional framework of HVRPTW. These constraints ensure that a vessel may only be assigned to a berth with sufficient spatial dimensions. Constraint (7) defines the handling time for vessel i, depending on the berth to which it is assigned. The handling time is influenced by several parameters: vessel i's cargo quantity Q_i, the time T required for a single crane to handle one unit of cargo, the number N^j of operational cranes at berth j, and the maximum number of cranes that vessel i may utilize. The effective number of cranes used is the minimum of the mooring berth's available cranes and the vessel's crane requirement. Thus, the potential handling time of vessel i at some berth j should be proportional to its

cargo divided by this effective number of cranes of j that could actually be utilized for i's cargo, scaled by T. All this information is precomputed per vessel-berth pair (i, j) and stored by the input parameter H_i^j, which is the ceiling (to preserve integrity of the input data) of the previously mentioned computation. The *temporal* constraints (8) and (9) enforce the (hard) time window constraints; each vessel i that is not denied service should be served by some berth within the predefined time window $[T_i^{ea}, T_i^{rd} + T^{dp}]$. In particular, Constraint (8) ensures that i's service cannot begin before its estimated time of arrival (T_i^{ea}). Constraint (9) dictates that i's service-completion time at berth j, $\boldsymbol{t_i^j} + \boldsymbol{h_i}$, must not exceed i's latest allowable departure time $T_i^{rd} + T^{dp}$. Constraint (10) prevents simultaneous servicing of two different vessels at the same berth. M is a large positive integer introduced to linearize the otherwise non-linear constraint. Finally, Constraint (11) defines the departure delay ($\boldsymbol{t_i^{ld}}$), which is equal to the maximum of 0 and i's actual departure time (i.e., estimated time of arrival plus handling time) from j minus its requested time of departure.

As an example, consider the dataset 10-5 constructed by our data generation process (cf. Section 4.1), consisting of 10 vessels and 5 berths. Figure 3a shows the corresponding flow graph G, containing a 10-clique at its centre, with its edges depicted in orange. The origin o has out-degree 11, while the destination d has in-degree 11. Finally, the "idle-subtour" path is represented by the black edge (o, d). The optimal solution produced by the exact MILP solver indicates the subtours followed by the 5 berths, as shown in Figure 3b.



(a) **(b)**

Figure 3 Flow graph for the data set D10-5. **(a)** The initial flow-graph $G = (V, E)$. **(b)** The optimal solution for the MILP instance, returned by the Gurobi exact solver. *Berth 1*: (o, 8, 9, d), *Berth 2*: (o, 3, 7, d), *Berth 3*: (o, 2, 0, d), *Berth 4*: (o, 6, d), *Berth 5*: (o, 1, 4, 5, d).

3 Heuristics, Improvement Heuristics & Metaheuristics

Although generic (such as Gurobi or CPLEX) exact MILP solvers can be used to solve instances of DDBASPTW which are formulated as shown in Figure 2, the NP-hardness [8] of the problem renders this approach rather prohibitive even for relatively small instances. In real-world scenarios where timely decision-making is critical, waiting for an exact solution may not be acceptable. Instead, high-quality solutions produced by constructive and improvement heuristics or by metaheuristics, which require notably less computational time, are often preferred. These approaches sacrifice (smoothly) the optimality guarantee in favor of efficiency, making them suitable for addressing large-scale instances effectively. In recent years, much effort has been made in developing efficient constructive heuristics for VRP [9]. Improvement heuristics can also be applied after the construction of an initial solution, to further enhance solution quality through local optimization techniques. In this section we demonstrate how some well-known VRP heuristic techniques can be adapted for, and applied to DDBASPTW.

3.1 Constructive Heuristics

Constructive heuristics for VRP build routing solutions incrementally, starting with an empty solution and repeatedly extending it, following predefined empirical rules. In our context, these heuristics produce the subtours for the berths (which are perceived as servicing vehicles) as they service the vessels (corresponding to customers to be served). For each berth, a subtour is determined as a sequence of vessels assigned to it, representing the order by which these particular vessels are moored to and then served by the berth. The appropriate temporal constraints are then easy to check along this subtour. In particular, if vessel i is assigned to berth j with mooring time $\boldsymbol{t_i^j}$, no other vessel can be assigned to j during the interval $[\boldsymbol{t_i^j}, \boldsymbol{t_i^j} + \boldsymbol{h_i}]$, where $\boldsymbol{h_i}$ represents the handling time required for i. Vessel i is considered compatible with berth j if all the relevant spatial and temporal constraints are preserved: $L_i \leq L^j$, $D_i \leq D^j$, and $\boldsymbol{t_i^j} \in \left[T_i^{ea}, T_i^{rd} + T^{dp} - \boldsymbol{h_i}\right]$.

Nearest Neighbour. *Nearest Neighbour* (NN), is the simplest constructive heuristic for VRP instances. The core idea of NN is to iteratively extend existing subtours by appending unassigned vessels to the ends of the subtours of the *nearest* berths which are eligible for them. As a distance metric between a vessel and a berth's subtour, we consider the marginal increase in cost resulting from the vessel's insertion at the current endpoint of the berth's subtour, that is, the difference in the subtour's objective value before and after the assignment. Initially, all subtours are empty, and the heuristic proceeds by appending vessels to the subtours of eligible berths for them, until all vessels have been assigned or there are no eligible berths for any of the remaining unassigned vessels.

In the sequential variant of NN, the berth subtours are constructed one by one, expanding each time the current berth's subtour with the nearest compatible vessel that is still unassigned, until it cannot accommodate any more vessels, before proceeding to the next berth. In the parallel variant of NN, which is the one implemented in this work, all berth subtours are constructed simultaneously. In each iteration, each berth's subtour is extended with the closest compatible unassigned customer, which means that at most $|B|$ customers will be added. The parallel version often achieves better berth utilization, as it avoids the imbalance typically seen in the sequential version, where the last berth tends to service fewer vessels.

Finally, we consider this (parallel) variant of NN in two different scenarios with respect to the revelation of vessel arrivals. The first scenario, tagged as the *complete knowledge Nearest Neighbour* heuristic (ck-NN), exploits the a priori knowledge of all vessel arrivals. In each iteration it attempts to expand the subtour of every berth once. For each berth, the heuristic appends the *nearest* unassigned vessel. As previously mentioned, the *nearest* unassigned vessel to a berth is the one that results in the smallest marginal increase in total cost when appended to the end of its subtour. This process repeats in a round-robin fashion among the berths, until all vessels have been assigned or there are no other feasible assignments. The second scenario, tagged as the *predetermined order Nearest Neighbour* heuristic (po-NN), aims to adapt NN for online situations where there is no prior knowledge of all arrivals of vessels, but they are revealed to the heuristic one by one. Therefore, in each iteration of po-NN, only one vessel is processed, according to a predetermined order, and is assigned to the nearest eligible berth. Naturally, the order in which vessels are processed can influence significantly the quality of the resulting solution. In this work we focus on the most natural order, indicated by the arrival times of the vessels.

Insertion. *Insertion* (INS) is a constructive heuristic that, unlike the NN method, evaluates all possible positions within each subtour to identify the optimal insertion point of some unassigned vessel, without changing the relative order of the already assigned vessels in the

subtours. By considering all possible positions, the mooring times of already assigned vessels may be adjusted, provided they still satisfy all the relevant temporal constraints. As in the case of NN, two variants of INS are implemented, *predetermined order Insertion* (po-INS) which processes and allocates vessels in a predetermined order, and *complete knowledge Insertion* (ck-INS) which leverages full knowledge of all vessels. The po-INS variant begins by initializing an empty subtour for each berth and, in each iteration, inserts the next unassigned vessel according to the predetermined order, into an eligible subtour. After evaluating all the marginal cost increases for inserting the vessel to the cheapest position of each subtour, the subtour with the smallest marginal cost is eventually chosen for the insertion of the new vessel. The marginal cost of assigning vessel i to berth j is defined as the sum of i's actual waiting time $\boldsymbol{t_i^j} - T_i^{ea}$, its handling time $\boldsymbol{h_i}$, and its potential departure delay $\boldsymbol{t_i^{ld}}$. In addition, it includes the cumulative increases in waiting times and departure delays for other vessels k already scheduled at berth j after i, due to shifts in their mooring times $\boldsymbol{t_k^j}$ caused by inserting i into j's subtour. As already mentioned, in this work we consider that the vessels are revealed according to their increasing estimated times of arrivals. In contrast, ck-INS operates in $|S|$ iterations, where each iteration evaluates all possible insertion positions for every unassigned vessel in each berth, always selecting the insertion that results in the lowest increase of the overall cost.

Quick and Dirty Insertion. The *quick and dirty Insertion* heuristic (qd-INS) is a restriction of INS, designed to compute even more efficiently a feasible solution, while maintaining an acceptable level of solution quality. The key distinction from INS lies in the following restriction: when inserting a new vessel, not only the relative order, but also the actual mooring times of all the previously assigned vessels to subtours cannot be altered. In other words, this is a non-preemptive variant of the INS heuristic (exactly like the NN heuristic), since it does not affect at all the assignments of previously allocated vessels. As a result, only insertion positions that leave existing mooring times unchanged are considered eligible. This *non-preemption* limitation reduces the number of eligible insertion positions, thereby speeding up the process. The algorithm evaluates all eligible positions across all subtours and selects the one that minimizes the additional cost for the new vessel. This time, the additional cost is defined solely by the new vessel i's waiting time $\boldsymbol{t_i^j} - T_i^{ea}$, its handling time $\boldsymbol{h_i}$, and its potential departure delay $\boldsymbol{t_i^{ld}}$. Once more, the order in which vessels are processed can significantly influence the final outcome. When the vessels are handled in a predetermined order, the corresponding variant of qd-INS is referred to as *predetermined order quick and dirty Insertion* (po-qd-INS). The *complete knowledge quick and dirty Insertion* (ck-qd-INS) is the variant of qd-INS that leverages full vessel information in advance.

3.2 Improvement Heuristics

Improvement heuristics explore the neighbourhood of the incumbent solution to achieve improvements in the objective value in a stronger preemptive manner that allows not only the temporal relocation of a vessel to a different slot of the same berth, but also its reassignment to a different berth. They typically converge quickly to a locally optimal solution w.r.t. the particular governing rule that determines the candidate berths and vessel servicing positions, and thus are efficient at solving large-scale routing problems [9]. In this section, we demonstrate how some well-known VRP improvement heuristics can be adapted to DDBASPTW.

Reinsert. The *Reinsert* improvement heuristic iteratively refines a given solution by selectively removing vessels from an existing schedule and then trying to reinsert them into alternative berths, in hope of improving the overall objective value. In each round, the algorithm performs a full pass over all vessels in the current schedule. For each vessel, it evaluates the effect of removing it from its assigned berth and attempting an insertion into an alternative eligible berth and berthing position, respecting all the vessel-to-berth eligibility constraints. If such a reinsertion leads to an improved solution, then the incumbent solution is immediately updated and the algorithm proceeds to the next vessel. The process continues until a full pass over all vessels fails to discover an improved solution.

Swap. The *Swap* improvement heuristic explores the neighbourhood of the current solution by evaluating all feasible pairwise exchanges of vessel assignments between two berths. Specifically, it generates all valid vessel pairs whose berth assignments could potentially be swapped without violating any vessel-to-berth eligibility constraint. For each candidate pair, the algorithm attempts to insert each vessel into the berth originally assigned to the other. If both these insertions are feasible and the resulting solution improves the overall objective value, the swap is immediately accepted and the current solution is updated; otherwise, the incumbent solution remains unchanged. The algorithm then proceeds to evaluate the next pair. By systematically exploring all such pairwise exchanges, the heuristic efficiently identifies and applies locally improving moves that may reduce the overall cost.

3.3 Metaheuristics

Heuristics provide a fast and practical way to either construct incrementally a solution or improve an existing solution, but they often come with limitations. Their design typically focuses on local optimization, which naturally leads to suboptimal global solutions. Additionally, many heuristics are sensitive to input order or initial conditions, which may affect their consistency and effectiveness. To address these challenges, metaheuristic approaches are employed. Metaheuristics provide high-level algorithm principles [6] that are less problem-dependent and incorporate concepts inspired by natural phenomena or physical processes. By leveraging these principles, a metaheuristic facilitates a more thorough exploration of the solution space, improving both the quality and the robustness of the resulting solutions. Metaheuristics are typically categorized into two classes: (i) *single-solution methods* which iteratively modify and improve a single candidate solution; and (ii) *population-based methods*, which operate on, and evolve a set of solutions simultaneously. In this section, one single-solution metaheuristic and two population-based metaheuristics are presented.

Cuckoo Search Algorithm. The *Cuckoo Search Algorithm* (CSA) [19] is a population-based metaheuristic optimization algorithm inspired by the breeding behavior of certain cuckoo species. Some cuckoos lay their eggs in the nests of other host birds. Hosts that discover these foreign eggs may either discard them or abandon the nest entirely. Inspired by this reproductive strategy, the following three rules are implemented to apply CSA to optimization problems: (i) Each cuckoo lays one egg at a time in a randomly selected nest; (ii) the best nests, containing eggs of high quality, are preserved and carried forward to the next iteration; (iii) the number of host nests is fixed and cuckoo eggs are discovered by host birds with a probability $p_{csa} \in (0, 1)$.

While CSA is rarely applied to VRP variants, its simplicity and ease of implementation make it a useful choice for experimentation and as a benchmarking baseline. CSA is adjusted for DDBASPTW as follows (cf. Algorithm 1): A nest is a set of unique assignments of

vessels for their servicing, where each vessel assignment is a pair of values (time, berth) that signifies the mooring time of the vessel at the particular berth. An egg represents either a berth or a mooring time for a particular vessel. The cuckoo egg represents a new assignment for a vessel (either a new berth, or a new mooring time). The total number of nests indicates the entire population of solutions. Each nest contains $2N$ eggs, i.e., two eggs (representing berth and mooring time) for each of the N vessels. In general, the goal of CSA is to employ cuckoo eggs for the exploration and replacement of nests with lower quality.

At first, m nests are created, forming the initial population, from a set of initial solutions which are generated using a slightly modified version of the po-INS heuristic; before processing the vessels, a shuffling step is applied, altering the order in which the vessels are processed. As a result, different vessel-allocation orders are produced for po-INS. In the initial population, the nest with the lowest cost (i.e., objective value) is identified. The reproduction step is carried out first: for each existing nest, a proportion p_v of vessels is randomly selected, and a *mutation* operator is applied to the solution for each of them. The *mutation* operator consists of moving the selected vessel to a new, randomly chosen berth. The vessels that remain in the previous berth rearrange their mooring times (but not their relative order) to reduce their waiting times and departure delays as much as possible. A check is performed next to determine if the mutant nest is better than the original nest, comparing their objective values. In that case, the mutant nest replaces the original one. Following this, the probabilistic event of the host discovering the cuckoo's egg is implemented. This probability, p_{csa}, ensures that in each iteration, approximately a fraction p_{csa} of the population is replaced by newly generated random solutions. These new solutions are once again provided by po-INS after shuffling the vessels. The nest with the lowest objective value in the current iteration is identified and compared with the best nest discovered so far. If its cost is lower than the currently best solution, it replaces it. The algorithm terminates when a predefined number of iterations is reached. CSA leverages the rapid execution of po-INS, which is called numerous times, making its speed essential for ensuring the overall computational efficiency of CSA.

Algorithm 1 Cuckoo Search Algorithm.

Input: Set of berths, set of n vessels
Output: x_{best}: best nest found

Initialize a population $\boldsymbol{X}$ of size m;
`/*` $f(x)$ `denotes the objective value of solution` x `*/`
$x_{best} \leftarrow$ find a nest (i.e., solution) with the lowest objective value $f(x)$ in population $\mathbf{X}$;
for $t \leftarrow 1$ **to** *number of iterations* **do**
  for $i \leftarrow 1$ **to** m **do**
    $x_{new} \leftarrow$ Perform mutation operation for $p_v \cdot n$ randomly selected vessels on $\boldsymbol{X}[i]$;
    if $f(x_{new}) < f(\boldsymbol{X}[i])$ **then** $\boldsymbol{X}[i] \leftarrow x_{new}$;
  end
  for $i \leftarrow 1$ **to** m **do**
    if $rand(0,1) \leq p_{csa}$ **then** $\boldsymbol{X}[i] \leftarrow$ newly generated solution;
  end
  $x_{localbest} \leftarrow$ Find nest with lowest objective value among the current nests of $\boldsymbol{X}$;
  if $f(x_{localbest}) < f(x_{best})$ **then** $x_{best} \leftarrow x_{localbest}$;
end
return x_{best};

Genetic Algorithm. The *Genetic Algorithm* (GA) is a well-known population-based metaheuristic which is inspired by a biological evolution process. GA mimics the Darwinian theory of survival of the fittest in nature. The variants of GA are commonly used to generate high quality solutions to VRP instances and BASP instances via biologically inspired operators such as *selection*, *crossover* and *mutation*. In the context of the GA, a solution, i.e., a complete assignment of all vessels (if possible) to berthing positions, is often referred to as a *chromosome*.

The variant of GA that is implemented in this work for DDBASPTW (cf. Algorithm 2) begins by generating an initial population of size m, following the same procedure that was described for CSA. It then proceeds with a predefined number of iterations. In each iteration, m pairs of "chromosomes" (i.e., solutions in the current population) are selected as parents for the offspring population using binary tournament selection. In a binary tournament, two chromosomes are randomly chosen, and the one with the better objective value is selected. After selecting m parent pairs, a crossover operation is applied to generate offspring; for each vessel present in the parent solutions, the offspring inherits the assignment from one of the two parents with equal probability (50%). If an inherited assignment conflicts with previously inherited ones, the vessel is temporarily excluded from the offspring and stored for later reinsertion. These unassigned vessels are subsequently reinserted using a po-INS based repair mechanism. Next, the mutation operator, exactly as described in the previous section for CSA, is applied to each offspring. Consequently, the newly generated population, consisting of the m offspring solutions, replaces entirely the parent population. At the end of each iteration, the best solution found so far is updated, whenever a better one exists in the current population. Upon completion of all iterations, the algorithm returns the best solution found.

Algorithm 2 Genetic Algorithm.

Input: Set of berths; Set of vessels; Number of iterations;
Output: x_{best} : best solution found

```
1  Initialize a population of size m, applying po-INS to randomly shuffled orders of the vessels;
   /* f(x) denotes the objective value of solution x                                   */
2  x_best ← The solution with the lowest objective value f(x) in the population;
3  for number-of-iterations rounds do
4  |  Select m pairs of chromosomes (i.e., solutions) from the population;
5  |  Apply a crossover operation to each pair of chromosomes, to create an offspring;
6  |  Apply mutation to each offspring;
7  |  Replace the population with newly generated population of offsprings;
8  |  x' ← The solution with the lowest objective value in the population;
9  |  if f(x') < f(x_best) then x_best ← x';
10 end
11 return x_best;
```

Adaptive Large Neighborhood Search. *Adaptive Large Neighborhood Search* (ALNS) is a single-solution metaheuristic which was first introduced by Stefan Ropke and David Pisinger [16] as an extension of the Large Neighborhood Search (LNS) metaheuristic. LNS iteratively destroys and repairs a solution by applying one *destroy* operator and subsequently a *repair* operator. The destroy operator removes parts of a solution while the repair operator reinserts the removed parts making it a complete solution. Similar to its preceding method, ALNS deploys the same sub-classes of operators: destroy and repair. A current solution will

be destroyed by the destroy operators and then repaired using the repair operators. The novelty of ALNS is that it now allows us to utilize multiple operators within the same searching process in an adaptive way, where this adaptiveness is attained by recording weights representing the effectiveness of each destroy-repair pair in producing better solutions and dynamically adjusting the random selection of destroy-repair methods proportionally to these weights [18]. ALNS is one of the most widely used metaheuristics for solving instances of VRP variants, and in this section, we adapt it to address DDBASPTW.

ALNS takes as input a feasible initial solution x, a set of destroy and a set of repair operators, and a parameter called *updatePeriod*. The algorithm returns the best solution found, denoted by x_{best}. The algorithm works as follows (cf. Algorithm 3): Initially, a temporary solution x is constructed and the weights and selection probabilities associated with each operator are uniform. The algorithm then proceeds iteratively for a fixed number of iterations. In each iteration, one destroy and one repair operator are selected using the roulette wheel mechanism: each operator is selected with a probability proportional to its weight, which reflects its effectiveness so far. These operators are then applied to modify the current solution. The resulting solution x' is evaluated and potentially accepted based on a *simulated annealing* criterion, which accepts x' unconditionally if it improves the objective value (i.e., $f(x') < f(x)$). Otherwise, it accepts x' with a probability $e^{-(f(x')-f(x))/T}$, where T is a *temperature* parameter. This mechanism allows the algorithm to escape from local optima by occasionally accepting worse solutions. The temperature T gradually decreases by performing the operation $T = aT$ in each updating period, using a *cooling rate* α, where $0 < \alpha < 1$, thereby reducing the likelihood of accepting inferior solutions over time. If the new solution improves upon the best solution found so far, it replaces x_{best}. Every *updatePeriod* iterations, the weights and selection probabilities of the operators are updated to reflect their effectiveness so far. An operator's effectiveness weight is evaluated based on how frequently it has led to an accepted solution based on the simulated annealing criterion, relative to the number of times it was applied. Once the termination condition is satisfied, the algorithm returns the best solution obtained.

Algorithm 3 Adaptive Large Neighborhood Search.

Input: A feasible solution x, destroy and repair operators, parameter *updatePeriod*
Output: x_{best} : best solution found

$x_{best} \leftarrow x$; $x' \leftarrow x$; $i \leftarrow 1$;
Initialize weights and probabilities;
for $t \leftarrow 1$ *to number of iterations* **do**
  Select a destroy and a repair operator;
  $x' \leftarrow$ repair(destroy(x));
  if *accept*(x', x) **then** $x \leftarrow x'$;
  /* $f(x)$ denotes the objective value of solution x */
  if $f(x) < f(x_{best})$ **then** $x_{best} \leftarrow x$;
  if $i =$ *updatePeriod* **then**
    Update weights and probabilities; $i \leftarrow 0$;
  end
  $i \leftarrow i + 1$;
end
return x_{best};

Five destroy operators were implemented to selectively dismantle parts of the current solution: (i) random removals of assigned vessels (nodes), (ii) clearance of randomly selected berths (subtours), (iii) removal of vessels that significantly contribute to the total objective cost (commonly referred to as *Worst Removal*), (iv) removal of vessels that are spatially

similar to a selected seed vessel (a special case of the well-known *Shaw Removal*, based on physical attributes such as length or draft) and (v) removal of vessels that are temporally similar to a selected seed vessel (also a special case of the Shaw Removal, based on mooring time). The extent of destruction is governed by a parameter *deg*, known as the *degree of destruction*, which controls the proportion of the solution to be removed. A sufficiently high value of *deg* allows the removal of structurally important parts of the solution, increasing the likelihood of escaping from local optima and improving the solution quality.

Six repair operators were implemented to reconstruct solutions after destruction. Three of these are based on po-qd-INS, differing only in the order in which the removed vessels are processed. Specifically, the orderings are determined by T_i^{ea}, T_i^{rd} and P_i (which can also be thought of as a measure of importance). The fourth repair operator leverages the po-INS heuristic, using the order in which the vessels were removed. The fifth operator repairs the solution using the ck-NN heuristic of constructing solutions. Finally, the sixth repair operator is based on a 2-regret-insertion, which iteratively selects and inserts a vessel that possesses the largest regret value in the list of removed (and still unassigned) vessels. The regret value of a vessel is obtained by taking the difference between the cost resulting from the best insertion position and the cost resulting from the second best insertion position.

3.4 Implementation Details

In our implementation, we used *interval trees* [14] to represent the subtour of each berth. An interval tree is a self-balancing binary tree that stores intervals in the format $[low, high]$. The left subtree contains intervals that precede the node's interval, while the right subtree contains those that follow. In this work, interval trees only represent feasible vessel-to-berth assignments, so all intervals are non-overlapping. They efficiently support insertion, deletion, interval searching, and overlap detection. Since each berth's subtour is defined by its occupancy periods, using interval trees allows efficient checks for availability within specific time intervals. By maintaining balance through rotations during updates, all operations – namely insertion, deletion, and overlap checking – run in $O(\log(n))$ time, where n is the number of stored intervals.

4 Experimental Evaluation

All experiments were conducted on an Ubuntu 22.04 PC equipped with an AMD EPYC 7552 48-Core Processor and 256 GB of RAM. The algorithms were implemented in C++, and the MILP models were solved using Gurobi 12.0.0 with default settings, utilizing 32 threads.

4.1 Data Generation

For our experimental evaluation, we used publicly available data on vessel arrivals and berthing activity from [12], complemented by a synthetic-data generator that produces instances resembling real-world conditions. At the port level, berth depths are generally standardized to accommodate a wide range of vessels. In our case, the port consists of three types of berths (A, B, and C) with respective depths of 9, 12, and 14 meters. Type A berths have lengths uniformly sampled from the range $[50, 170] \equiv 0(mod\ 5)$ meters and are equipped with a random number of cranes between 1 and 5. Type B berths have lengths in the range $[171, 230] \equiv 0(mod\ 5)$ meters and 2 to 6 cranes, while type C berths range from $[231, 350] \equiv 0(mod\ 5)$ meters in length and are randomly assigned between 2 and 8 cranes. As far as vessels are concerned, we allocated them to the three berth types, defining the range of lengths permissible for each type. Regarding vessel drafts, since our port cannot

accommodate vessels with a draft exceeding 14 meters, it is more realistic to employ a sigmoid curve with a plateau at 14 meters. Subsequently, the vessel drafts can be selected through a uniformly random ($\mathcal{R}$) selection around this value within a few units. In particular, vessel i's draft is determined as follows:

$$D_i = \mathcal{R}\big(\max\{1, \sigma(L_i) - 2\}, \min\{14, \sigma(L_i) + 2\}\big) \tag{12}$$

where $\sigma(L_i) = \dfrac{14}{1 + e^{-0.02 \cdot (L_i - 187.5)}}$. The maximum number of cranes that could be employed for servicing vessel i is determined according to the uniformly random selection $\mathcal{R}\Big(\max\{1, \lfloor L_i/50 \rfloor - 2\}, \lfloor L_i/50 \rfloor + 2\Big)$, where $\lfloor x \rfloor$ represents the integer part of a real number x. This formula ensures that the maximum number of cranes required for vessel i is proportional to its length, while maintaining variability within a reasonable range. The lower bound guarantees that at least one crane is required, even for smaller vessels. As the length of the vessel increases, more cranes could be employed to accommodate the larger volume of cargo and operational complexity. The cargo quantity carried by a vessel, measured in Twenty-feet Equivalent Units (TEU), is uniformly at random selected, depending on its length, as follows: (i) up to 100m: between 25 and 100 TEU; (ii) from 100 up to 150m: between 100 and 500 TEU; (iii) from 150 up to 200m: between 500 and 1250 TEU; (iv) from 200 up to 250m: between 1000 and 2250 TEU; (v) from 250 up to 300m: between 2000 and 3500 TEU; and (vi) over 300m: between 3000 and 5000 TEU. These intervals reflect the increasing cargo capacity of larger vessels, ensuring that the dataset maintains a realistic distribution of cargo loads. The random selection of cargo values within the specified ranges assures a controlled variability to the dataset while preserving logical consistency in the vessel size-to-cargo ratio.

Finally, each vessel $i \in S$ is associated with a penalty P_i, which is incurred only if the vessel is denied service by a solution. The penalty is designed to be sufficiently high to deter the exact MILP solver from leaving a vessel unserviced, except for situations where the problem would otherwise be infeasible. The penalty is defined as follows:

$$P_i = (T_i^{rd} + T^{dp} - T_i^{ea}) \cdot C_i^w + T^{dp} \cdot C_i^{ld} + h_i^{\text{worst}} \cdot C_i^h \tag{13}$$

where h_i^{worst} denotes the worst case handling time for vessel i, among all eligible berths.

4.2 Calibration of Parameters for Metaheuristic Algorithms

We specify the parameter values used in our metaheuristic implementations, which were calibrated through preliminary experiments towards both efficiency in the quality of the resulting solutions and fairness in their comparison with each other in the final experimental evaluation. For CSA, the population size m is 100, with 100 iterations, a mutation proportion p_v of 0.25, and discovery probability p_{csa} of 0.45. For GA, the population size and number of iterations are also set to 100. For ALNS, we use 1000 iterations and an *updatePeriod* of 50. The initial solution is generated via the po-INS heuristic. The simulated annealing acceptance criterion uses a dynamically initialized temperature based on operator performance in the first 5 iterations, a cooling ratio $a = 0.8$, and a destruction degree *deg* randomly chosen each iteration in the range $[0, 0.4]$ times the number of vessels.

4.3 Experimental Results

For the experimental evaluation, the cost parameters C_i^w, C_i^h, and C_i^{ld} are all set to 1 euro, and the time unit is standardized to 1 hour. For instance, if $h_i = 2$, vessel i requires 2 hours for being handled by the berth to which it is assigned, and contributes 2 euros to the objective function. The time required for a single crane to load or discharge one unit of cargo is set to $T = 5$ minutes $= (1/12)$ hours, and the delay policy is set to $T^{dp} = 10$ hours.

Results on Constructive Heuristics and Metaheuristics. Table 2 compares ten methods: Gurobi (GRB) (using a generic branch and cut algorithm), po-NN, ck-NN, po-INS, ck-INS, po-qd-INS, ck-qd-INS, CSA, GA, and ALNS. The comparison is based on computation times, optimality gaps, and number of vessels allocated across various data sets, all with a one-week planning horizon. The data set corresponding to each block of results is indicated in the first column, using the notation X-Y, where X denotes the number of vessels and Y the number of berths. For each such X-Y benchmark case size, we generated ten different random instances. Each random instance was solved twenty times to obtain stable runtime measurements. For heuristic methods, the execution time is the average over these twenty runs. For metaheuristic methods, both the execution time and the objective value are averaged over the twenty runs. The median across the ten random instances per X-Y size is then presented in the tables as follows. The "Time" row shows the *execution time* (in seconds). The "VA" row indicates the *number of assigned vessels* in each method's solution. The "Gap" row reflects the *relative error* from the reference value used for comparison. When GRB reaches an optimal solution (i.e., $Gap = 0$), the heuristic and metaheuristic methods are evaluated against this optimal value. If GRB fails to reach optimality within a three-hour limit for the execution time (i.e., $Gap > 0$), the comparison is made using a derived lower bound, computed from the objective value reported by GRB and the optimality gap it had reached at the time limit. In both cases, the relative error is computed as the difference between the method's objective value and the reference value, divided by the reference value. Instances where GRB hits the three-hour time limit are marked with an asterisk next to the execution time in the tables. Gap values in Table 2 are reported as decimals but represent percentages (e.g., a value of 0.2 corresponds to a 20% gap). Overall, the benchmark instances reflect realistic operational scales, with sizes larger or comparable to those of major terminals, such as the Piraeus Container Terminal [12] which can service up to 11 vessels simultaneously.

In terms of performance, the ck-NN heuristic successfully serves all vessels in 9 out of 13 benchmark cases, but optimality gaps range from 17% to 41% which are considered rather unsatisfactory. When not all vessels are assigned to berths, objective values increase due to the penalties applied for unallocated vessels. As expected, po-NN performs poorly across all benchmark cases, as it is a very naive approach. The po-INS heuristic assigns all vessels to berths in 12 out of 13 benchmark cases, failing only on benchmark case 40-10, where it assigns one fewer vessel than Gurobi. However, it is worth noting that this is an exceptionally difficult benchmark case, because even Gurobi manages to berth only 39 vessels within the three-hour limit.

For the benchmark cases where all vessels are successfully moored, po-INS achieves gaps ranging from 3.7% to 26.5%. In contrast, the performance of ck-INS is significantly worse, primarily due to its tendency to postpone the consideration of large vessels, which have high handling costs and fewer compatible berths, until later in the construction. By the time these vessels are considered, their feasible berths are often already occupied and the algorithm cannot reassign previously placed vessels or even reorder them, leading to denial of service for these vessels and, thus, high penalties in the objective value of the resulting solution. The po-qd-INS heuristic assigns all vessels to berths in 11 out of 13 benchmark cases with gaps varying from 5.7% to 27%. The ck-qd-INS variant also exhibits poor performance for the same reasons as ck-INS, as it similarly delays the consideration of larger vessels, leading to denial of service for many of them and thus incurring high penalties. Overall, we observe that po-NN, ck-INS, and ck-qd-INS yield unsatisfactory results and are excluded from further discussion. Among the remaining heuristics, it is evident that ck-NN performs the worst. Between po-INS and po-qd-INS, the former delivers higher solution quality, albeit with a

Table 2 Experimental results on Constructive Heuristics and Metaheuristics. In each scenario, the best running time and the smallest relative error are indicated by **blue-boldface** and **red-boldface**, respectively, among the three metaheuristics.

Data Set	Metric	NN		INS				CSA	GA	ALNS	GRB
		ck	po	po	ck	po-qd	ck-qd				
25-10	Time	0.00001	0.00001	0.00009	0.00058	0.00002	0.00008	0.129	0.047	**0.039**	10.6
	VA	25	25	25	24	25	22	25	25	25	25
	Gap	0.22	0.4	0.037	26	0.057	59	0.034	0.037	**0.02**	0
30-10	Time	0.00001	0.00001	0.00011	0.001	0.00002	0.00011	0.32	0.064	**0.059**	6.2
	VA	30	30	30	29	30	29	30	30	30	30
	Gap	0.17	0.3	0.065	41	0.065	41.2	0.007	0.027	**0**	0
35-10	Time	0.00001	0.00001	0.00013	0.00189	0.00003	0.00023	0.52	**0.085**	0.093	32
	VA	35	34	35	33	34	30	35	35	35	35
	Gap	0.18	8.4	0.12	34	8.14	85.1	0.029	0.026	**0.007**	0
40-10	Time	0.00002	0.00001	0.00016	0.002	0.00003	0.00027	0.64	0.12	**0.03**	101
	VA	36	36	38	35	38	31	38	39	38	39
	Gap	0.89	1.3	0.36	3.1	0.29	4.8	0.36	**0.006**	0.29	0
45-10	Time	0.00002	0.00001	0.0002	0.003	0.00004	0.0004	0.82	**0.09**	0.13	10800*
	VA	44	43	45	41	45	37	45	45	45	45
	Gap	19.96	7	0.18	49	0.24	94.9	0.164	0.169	**0.079**	0.042
60-20	Time	0.00003	0.00002	0.0003	0.005	0.00007	0.00074	1.04	**0.161**	0.38	10800*
	VA	60	60	60	57	60	55	60	60	60	60
	Gap	0.34	0.48	0.20	50	0.20	80.5	0.15	0.10	**0.09**	0.073
65-20	Time	0.00004	0.00002	0.00040	0.00858	0.00008	0.00096	1.98	**0.15**	0.55	10800*
	VA	65	65	65	63	65	60	65	65	65	65
	Gap	0.30	0.55	0.13	40	0.13	74.1	0.10	0.09	**0.07**	0.04
70-20	Time	0.00004	0.00002	0.00045	0.01216	0.00009	0.001	2.30	**0.19**	0.63	10800*
	VA	69	68	70	65	70	63	70	70	70	70
	Gap	3.35	9.85	0.20	60.4	0.21	73	0.14	0.12	**0.08**	0.04
75-20	Time	0.00004	0.00002	0.00055	0.01	0.0001	0.001	2.74	**0.21**	0.94	10800*
	VA	75	75	75	73	75	68	75	75	75	75
	Gap	0.41	0.52	0.265	30.3	0.265	81.9	0.21	0.20	**0.13**	0.06
80-20	Time	0.00004	0.00002	0.00058	0.014	0.0001	0.001	2.93	**0.22**	1.07	10800*
	VA	79	79	80	75	80	73	80	80	80	80
	Gap	5	12	0.16	59.86	0.19	77.4	0.13	0.12	**0.08**	0.07
85-20	Time	0.00008	0.00002	0.0006	0.019	0.0001	0.002	3.13	**0.29**	1.24	10800*
	VA	84	84	85	82	85	76	85	85	85	85
	Gap	11.9	13	0.10	30.59	0.11	65.6	0.14	0.11	**0.09**	0.08
90-20	Time	0.00005	0.00003	0.0007	0.02	0.0001	0.002	3.55	**0.30**	1.35	10800*
	VA	90	89	90	84	90	78	90	90	90	90
	Gap	0.4	6	0.191	60.52	0.195	109.5	0.114	0.121	**0.09**	0.07
100-25	Time	0.00006	0.00003	0.0009	0.045	0.0001	0.005	4.03	**0.37**	2.1	10800*
	VA	100	99	100	96	100	91	100	100	100	100
	Gap	0.33	2.2	0.08	31.2	0.09	56.2	0.058	0.054	**0.048**	0.034

slight increase in computational time. The po-qd-INS heuristic outperforms po-INS only in cases where both of them fail to allocate two vessels. Lastly, in four benchmark cases, namely, 30-10, 60-20, 65-20 and 75-20, po-INS and po-qd-INS produce identical objective values, which can be attributed to the predetermined vessel ordering used in both methods.

Regarding the metaheuristics, we observe that all three algorithms successfully allocate all vessels to berths in 12 out of 13 benchmark cases. In all cases, the metaheuristics consistently produce higher-quality solutions than the constructive heuristics, albeit with increased computational times. Among them, ALNS achieves the best performance, except for benchmark case 40-10, where only GA matches GRB in terms of the number of allocated vessels to berths. Overall, CSA and GA yield solutions of comparable quality, but GA is significantly faster. In summary, ALNS delivers the highest-quality solutions among all methods evaluated and, while slower than the constructive heuristics and the GA metaheuristic, it remains computationally efficient. Specifically, in the 12 benchmark cases where all vessels

are successfully allocated to berths, ALNS achieves optimality gaps ranging from 0% to 13% compared to GRB's gap which ranges from 0% to 8%, while being faster than GRB by factors ranging from 353 to 83,076.

Experimental Results on Improvement Heuristics. Table 3 presents the results of the improvement heuristics *Swap* and *Reinsert*, applied to the base heuristics ck-NN, po-INS, and po-qd-INS. For each benchmark case, the improvements were applied to all 10 instances and the median values across these instances are reported. The "Before" columns report the gap achieved by each base heuristic prior to applying the improvement heuristics (identical to the values in Table 2). The "Time" columns indicate the execution time in seconds per improvement heuristic, while the "Gap" columns show the updated gaps after applying *Swap* and *Reinsert*, respectively. We report results only for po-NN, po-INS, and po-qd-INS, as these were shown to be the best-performing constructive heuristics in Table 2.

Swap significantly enhances solution quality, particularly for the ck-NN heuristic which typically produces initial solutions with substantial room for improvement. In contrast, in benchmark cases such as 25-10 and 30-10, where po-INS and po-qd-INS already produce almost optimal solutions, the Swap heuristic offers little or no improvement. Similarly, in cases where some vessels remain unallocated, the Swap heuristic has limited impact, as the penalty for unassigned vessels tends to dominate the overall objective value. The Reinsert heuristic exhibits similar behaviour, proving especially effective when applied to ck-NN solutions for the same reasons. Again, when solutions are almost optimal, or when unassigned vessels drive the objective, Reinsert contributes minimally to the solution quality. When comparing Swap and Reinsert, there is no clear winner; both perform comparably and provide improvements when the solution space allows for meaningful adjustments. When benchmarked against metaheuristics, both improvement heuristics are faster but generally yield solutions of lower quality than CSA and GA and consistently worse than ALNS. Overall, when the initial solution is sufficiently suboptimal, both Swap and Reinsert effectively enhance solution quality, therefore standing between constructive heuristics and metaheuristics in the Pareto frontiers of the solution-quality to execution-time diagrams.

5 Conclusions and Future Work

In this work, we introduced a novel mathematical formulation of DDBASPTW, modeling it as an instance of HVRPTW. We then adapted two VRP constructive heuristics to the DDBASPTW, namely, Nearest Neighbour (NN) and Insertion (INS), and proposed qd-INS, a fast "quick-and-dirty" variant of INS. For all these heuristics, we considered both the online scenario and the case in which all vessel information is known in advance. Furthermore, we incorporated two improvement heuristics, Reinsert and Swap, both drawn again from the VRP literature, and applied one single-solution metaheuristic (ALNS) as well as two population-based metaheuristics (CSA and GA).

Our experimental results demonstrated that the VRP-inspired techniques adapted for DDBASPTW achieve satisfactory performances. In particular, ALNS exhibited outstanding results, producing solutions close to those obtained by exact solvers, while requiring significantly less computational time, making it a highly effective choice for real-time applications. Future work will focus on further exploring the connection between variants of BASP and VRP, with the aim of developing more advanced and specialized solution methods for BASP. Additionally, we plan to investigate alternative metaheuristic techniques to further enhance performance and address increasingly complex and dynamic problem settings.

Table 3 Experimental results on Improvement Heuristics.

Alg.	Before	Swap		Reinsert	
	Gap	Time	Gap	Time	Gap
25-10					
ck-NN	0.22	0.0003	0.16	0.0005	0.05
po-INS	0.037	0.0004	0.037	0.0002	0.037
po-qd-INS	0.057	0.0004	0.037	0.0002	0.057
30-10					
ck-NN	0.17	0.0006	0.045	0.0005	0.10
po-INS	0.065	0.0009	0.064	0.0004	0.065
po-qd-INS	0.065	0.0008	0.064	0.0003	0.065
35-10					
ck-NN	0.18	0.0012	0.0849	0.0015	0.04
po-INS	0.12	0.0018	0.07	0.0013	0.05
po-qd-INS	8.14	0.0015	8.06	0.0007	8.06
40-10					
ck-NN	0.89	0.0011	0.88	0.0007	0.88
po-INS	0.36	0.0018	0.35	0.001	0.35
po-qd-INS	0.29	0.0017	0.28	0.0008	0.28
45-10					
ck-NN	19.96	0.0017	19.85	0.0016	19.85
po-INS	0.18	0.0026	0.16	0.0013	0.1
po-qd-INS	0.24	0.0021	0.14	0.0011	0.18
60-20					
ck-NN	0.34	0.0035	0.20	0.0022	0.25
po-INS	0.20	0.0044	0.18	0.0024	0.16
po-qd-INS	0.20	0.0040	0.18	0.0022	0.17
65-20					
ck-NN	0.30	0.0039	0.20	0.0049	0.20
po-INS	0.13	0.0052	0.12	0.0031	0.10
po-qd-INS	0.13	0.0049	0.12	0.0039	0.11

Alg.	Before	Swap		Reinsert	
	Gap	Time	Gap	Time	Gap
70-20					
ck-NN	3.35	0.0070	3.25	0.0043	3.28
po-INS	0.2	0.0105	0.16	0.0035	0.11
po-qd-INS	0.21	0.0105	0.14	0.0031	0.08
75-20					
ck-NN	0.41	0.007	0.17	0.004	0.30
po-INS	0.265	0.008	0.22	0.005	0.21
po-qd-INS	0.265	0.008	0.23	0.003	0.21
80-20					
ck-NN	5.00	0.008	4.90	0.006	14.93
po-INS	0.16	0.010	0.14	0.004	0.12
po-qd-INS	0.19	0.009	0.15	0.005	0.13
85-20					
ck-NN	11.9	0.012	11.78	0.011	11.84
po-INS	0.10	0.014	0.08	0.005	0.09
po-qd-INS	0.11	0.013	0.08	0.004	0.09
90-20					
ck-NN	0.4	0.014	0.28	0.005	0.29
po-INS	0.191	0.020	0.182	0.006	0.187
po-qd-INS	0.195	0.020	0.189	0.005	0.19
100-25					
ck-NN	0.33	0.024	0.117	0.020	0.193
po-INS	0.08	0.028	0.07	0.007	0.07
po-qd-INS	0.09	0.027	0.08	0.007	0.075

References

1 Sheraz Aslam, Michalis Michaelides, and Herodotos Herodotou. Optimizing multi-quay berth allocation using the cuckoo search algorithm. In *Optimizing Multi-Quay Berth Allocation using the Cuckoo Search Algorithm*, March 2022. doi:10.5220/0011081200003191.

2 Sheraz Aslam, Michalis P. Michaelides, and Herodotos Herodotou. Enhanced berth allocation using the cuckoo search algorithm. *SN Comput. Sci.*, 3(4):325, 2022. doi:10.1007/S42979-022-01211-Z.

3 Katja Buhrkal, Sara Zuglian, Stefan Ropke, Jesper Larsen, and Richard Lusby. Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E: Logistics and Transportation Review*, 47:461–473, July 2011. doi:10.1016/j.tre.2010.11.016.

4 Juan F. Correcher, Federico Perea, and Ramon Alvarez-Valdes. The berth allocation and quay crane assignment problem with crane travel and setup times. *Computers & Operations Research*, 162:106468, 2024. doi:10.1016/j.cor.2023.106468.

5 Maxim A. Dulebenets, Masoud Kavoosi, Olumide Abioye, and Junayed Pasha. A self-adaptive evolutionary algorithm for the berth scheduling problem: Towards efficient parameter control. *Algorithms*, 11(7), 2018. doi:10.3390/a11070100.

6 Michel Gendreau, Jean-Yves Potvin, et al. *Handbook of metaheuristics*, volume 2. Springer, 2010.

7 Bokang Li, Zeinab Elmi, Ashley Manske, Edwina Jacobs, Yui-yip Lau, Qiong Chen, and Maxim A. Dulebenets. Berth allocation and scheduling at marine container terminals: A state-of-the-art review of solution approaches and relevant scheduling attributes. *J. Comput. Des. Eng.*, 10(4):1707–1735, 2023. doi:10.1093/JCDE/QWAD075.

8 Andrew Lim. The berth planning problem. *Operations Research Letters*, 22(2):105–110, 1998. doi:10.1016/S0167-6377(98)00010-8.

9 Fei Liu, Chengyu Lu, Lin Gui, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Heuristics for vehicle routing problem: A survey and recent advances. *CoRR*, abs/2303.04147, 2023. doi:10.48550/arXiv.2303.04147.

10 United Nations Conference on Trade and Development (UNCTAD). Handbook of statistics, 2023.

11 World Trade Organization. World trade statistical review, 2023.

12 Piraeus Container Terminal S.A. *Piraeus Container Terminal*, 2025. URL: https://www.pct.com.gr.

13 Luigi Pio Prencipe and Mario Marinelli. A novel mathematical formulation for solving the dynamic and discrete berth allocation problem by using the bee colony optimisation algorithm. *Applied Intelligence*, 51(7):4127–4142, July 2021. doi:10.1007/S10489-020-02062-Y.

14 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

15 Lingyu Ran, Boning Liu, Guiqing Zhang, and Yongxi Cheng. An improved particle swarm optimization algorithm for berth allocation and time-variant quay crane scheduling problem during an emergency. *Expert Syst. Appl.*, 269:126406, 2025. doi:10.1016/J.ESWA.2025.126406.

16 Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, November 2006. doi:10.1287/trsc.1050.0135.

17 Paolo Toth, Daniele Vigo, Paolo Toth, and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, USA, 2014.

18 Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146:105903, 2022. doi:10.1016/j.cor.2022.105903.

19 Xin-She Yang and Suash Deb. Cuckoo search via levy flights, 2010. arXiv:1003.1594.