Constraint Programming and Local Search Heuristic: A Matheuristic Approach for Routing and Scheduling Feeder Vessels in Multi-Terminal Ports

David Sacramento, Christine Solnon, David Pisinger

Constraint Programming and Local Search Heuristic: A Math-heuristic Approach for Routing and Scheduling Feeder Vessels in Multi-Terminal Ports

David Sacramento*[1], Christine Solnon[2], and David Pisinger[1]

[1]*DTU Management - Technical University of Denmark, Kongens Lyngby, Denmark*
[2]*INSA Lyon, CITI, INRIA CHROMA, F-69621 Villeurbanne, France*

Abstract

In the liner shipping business, shipping ports represent the main nodes in the maritime transportation network. These ports have a collection of terminals where container vessels can load and discharge containers. However, the logistics and planning of operations differ depending on the vessel size. Large container vessels visit a single terminal, whereas smaller container vessels, or feeder vessels, visit several terminals to transport all containers within the multiple terminals of the port. In this paper, we study the Port Scheduling Problem, the problem of scheduling the operations of feeder vessels in multi-terminal ports. The resulting problem can be identified as a version of the General Shop Scheduling Problem. We consider a Constraint Programming formulation of the problem, and we propose a math-heuristic solution approach for solving large instances. The proposed math-heuristic is a hybrid solution method that combines Constraint Programming with a local search heuristic. The solution approach benefits from the fast search capabilities of local search heuristics to explore the solution space using an Adaptive Large Neighbourhood Search heuristic. During the search, we further use the Constraint Programming model as an intensification technique, every time a new best-known solution is found. We conduct detailed computational experiments on the *PortLib* instances, showing that the incorporation of Constraint Programming within the heuristic search can result in significant benefits. The high instability in solution quality obtained by local search heuristics can be lowered by a simple combination of both methods.

Keywords: Feeder Vessels, Maritime Optimisation, Constraint Programming, ALNS, Math-heuristic

1 Introduction

Maritime transportation is one of the cheapest and most efficient mode of transportation, carrying out more than 90% in volume of the international trade [38]. The main characteristic of seaborne transportation is its huge capacity, which allow to transport large volumes over long distances. This makes maritime transportation one of the best options for international trade.

*Corresponding Author.
Email addresses: dsle@dtu.dk (David Sacramento), christine.solnon@liris.cnrs.fr (Christine Solnon), and dapi@dtu.dk (David Pisinger).

Within the maritime industry, we focus on the liner shipping industry. Here, container vessels transport all of their cargo in standard containers, which are commonly given in two standard sizes: *twenty foot equivalent units* (TEU) and *forty foot equivalent units* (FFE). This standardisation helps to add flexibility and versatility in the logistics operations, since containers can be used across different modes of transportation. Due to the steady increase in the containerised cargo over the past 10 years, the size of current vessels is ever increasing, and vessels that can carry over 20,000 TEUs are already sailing the seas [37].

The liner shipping industry is built around so called *services*, operating in a similar way to public transportation services, such as bus services. In liner shipping, a service is a cyclic itinerary of port-visits, sailed by a number of similar vessels with a weekly frequency. If the total round trip of a service takes 8 weeks to complete, then 8 vessels of the same class are deployed in the service to ensure weekly departures from the ports. The shipping companies, also called carriers, operate the vessels, and the largest carriers operate over 600 vessels.

The liner shipping network covers most of the seas and oceans around the world. However, to facilitate logistics and cargo transportation, each region is divided between a few large ports, called *hubs*, and many small ports, called *feeder ports*. Liner services are usually operated by large vessels, also called *liner vessels*, and these vessels only visit hub ports in each region, where they discharge and load all the containers designated to the corresponding region. Then, smaller vessels, called *feeder vessels*, transport the containers from the hubs to the feeder ports.

Generally, shipping ports have a collection of container terminals, which are designated areas within the port where vessels can load and discharge containers by using quay cranes. Feeder ports are usually small, and they tend to have a single container terminal. Large ports, such as Rotterdam or Singapore, are multi-terminal ports. Here, the terminals are distributed along the port area, and they tend to be far apart with a significant sailing distance between them. When planning the port visits of container vessels in multi-terminal ports, the logistics and planning of operations are different depending on the vessel size. Liner vessels have large carrying capacities, and they normally visit only a single terminal in the port. At this terminal, the liner vessel discharges and loads all the containers designated to the port. Moreover, as liner vessels have priority when planning the port visits, the terminal is assigned long time in advance. On the other hand, feeder vessels usually handle cargo from multiple liner vessels, and they need to visit several terminals to transport all containers to their corresponding destinations.

This paper studies the problem of scheduling the operations of feeder vessels in multi-terminal ports. This problem was first introduced in our previous work in Hellsten et al. [15], where we defined the *Port Scheduling Problem* (PSP). In this paper, we consider a *Constraint Programming* (CP) formulation of the same problem. Due to the extensive library of variables and constraints within the CP tools for solving optimisation problems, the CP formulation can easily accommodate more flexible definitions of decision variables and all additional practical constraints. One of the biggest advantages of CP compared to linear programming is the possibility of modelling non-linear constraints in a straightforward way without the burden of transforming them into linear constraints (using *big-M constraints*, for example). It is experimentally shown that CP can return high-quality solutions in short computational times. Furthermore, we present an *Adaptive Large Neighbourhood Search* (ALNS) math-heuristic for solving large instances. A

math-heuristic is an optimisation algorithm made by the interoperation of metaheuristics and mathematical programming techniques, combining the strengths of mathematical programming methods to intensify the search and the strengths of metaheuristics to exploit a large solution space [5]. In the math-heuristic proposed in this paper, we explore the solution space using an adapted version of the ALNS heuristic from [15]. The main adaptions of the ALNS heuristic are described in Section 4.1. Moreover, we use CP to intensify the search every time a new best-known solution is found. It is further shown in the computational experiments that the combination of both methods provides good results within reasonable computation times, improving the performance of the former ALNS heuristic. We demonstrate that the obtained results are competitive with metaheuristic methods, showing that CP methods can be successfully applied to scheduling problems in the maritime sector.

1.1 Contribution and Overview of the Paper

The main contribution of the paper is the ALNS math-heuristic framework for the PSP, which combines the fast search capabilities of local search heuristics with the systematic search of CP to explore the solution space. In order to develop the math-heuristic, we first need to consider a CP formulation of the problem. The CP formulation provides a flexible modelling framework, allowing a more compact and straightforward definition of the problem. Additionally, the CP formulation can easily add new side constraints to the problem without changing the overall structure of the model. We compare the performance of different solution methods on the *PortLib* instances, and show that the ALNS math-heuristic reports superior results for large-sized instances. Furthermore, we conduct detailed computational experiments to analyse the performance of the math-heuristic under different configurations, providing insights on how to design the math-heuristic framework for the PSP.

The remainder of this paper is organised as follows. In Section 2, we formally describe the PSP and present a review of the literature related to this problem. Section 3 presents how the PSP can be modelled using CP. The section further provides an experimental evaluation of the performance of CP with different search parameterisation, and compares CP with other mathematical optimisation programs. Section 4 describes the hybrid solution method which combines CP with an ALNS heuristic. A comparison on the performance of the solution method against different configurations is further discussed in this section. Finally, Section 5 summarises and concludes the paper.

2 Problem Definition

We study the operational planning process in an international shipping company. The carrier wants to schedule all the operations for a set of feeder container vessels visiting a multi-terminal port while respecting a variety of practical constraints. In the following, we briefly describe the main characteristics of the problem. For a more thorough definition of the problem, we refer the reader to the seminal paper [15].

Each feeder vessel has a number of operations to perform in the port. These operations are fixed, and most of them have to be carried out at different terminals. Generally, these operations consist in

discharging or loading all containers designated to the specific terminal. However, other operations such as maintenance or repair operations may be also considered. This means that a feeder vessel can have multiple operations assigned to the same terminal, although all of these operations may not be required to be performed during a single visit to the terminal. In this work, we only assume handling operations, i.e. discharge and loading operations. We consider non-preemptive operations, meaning that an operation cannot be interrupted once started. Each operation has a required service time, which is usually calculated as the number of containers to be handled at the terminal divided by the historical data on the gross crane productivity of the terminal.

There is a given arrival time and estimated latest departure time for each feeder vessel visiting the port, and all operations assigned to a vessel must be completed within this interval. Some operations may be subject to stricter time windows, in order to ensure the transshipment of cargo between services. The time window of an operation indicates a time interval in which the operation can be performed. However, vessels can wait in nearby lay-by terminals until the opening of the time window. Each vessel arrives to the port with an initial cargo capacity, and has a maximum cargo capacity that must be respected at any time during the port stay. Additionally, there may be precedence relations between some operations which have to be respected. The most common relation arises from the imposition for a vessel to discharge all the containers in a terminal before loading containers at the same terminal. Other precedence relations may be defined by the transshipment of cargo between feeder vessels, or by stowage plans. Lastly, there is a non-negligible sailing distance between the terminals within the port.

We assume that each terminal can serve at most one vessel at a time. This comes from the fact that the carrier must negotiate in advance with the terminals a number of berths to serve the carrier's fleet. Here, a berth provides quay cranes and workforce to serve a single feeder vessel during a certain time period in the terminal. During the previously negotiated time periods, the terminal is freely available to serve the carrier's fleet, and the carrier must decide *when* and *which* feeder vessel should visit the terminal during those time periods so that all operations can be carried out and all containers can be transferred. Although it is of course possible to serve multiple vessels during a single berth, this would require further negotiations with the terminals, which may result in greater leasing costs and lower terminal productivity. Therefore, the carrier prefers to avoid multiple berths at the same time at terminals, and this is therefore a hard constraint in our model. Furthermore, we assume that terminals can also have time windows, corresponding to time periods where a terminal is closed or not available for the carrier in question. A visual representation of a solution of the problem is depicted in Figure 1.

The *objective* is to define an operational schedule for feeder vessels which minimise the total staying time in the port while respecting all the aforementioned constraints. There are two goals with the objective function: (1) We minimise the weighted starting time of operations, scheduling operations as early as possible in order to design a 'compact schedule'. This leads to more robust schedules leaving more slacks for future changes; and (2) we minimise the departure time of vessels from the port. We can observe an important trade-off between the two objective functions, as scheduling operations early will consequently ensure the early departure from the port. However, the minimisation of departure times has a higher

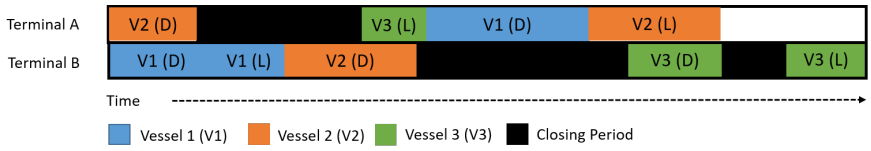

Figure 1: *Example of an operational schedule for the PSP with three feeder vessels, two container terminals and three closing periods. Feeder vessels are depicted with different colors, and closing periods are depicted in black. The operations are represented by rectangles, whose length is given by the service time and color corresponds to the associated vessel. The vessel name and type of operation (D: Discharge, L: Loading) are written within the rectangle.*

impact on the solution, as feeder vessels can benefit from *slow-steaming* by sailing and arriving earlier to the next port. The reason to consider a time-minimisation objective function is that most of the monetary costs in this problem are either fixed or marginals.

2.1 Literature Review

Maritime logistics is a complex process, which involves the collaboration and coordination of many sub-problems. In order to design a competitive and efficient maritime shipping network, port operations must be carried out smoothly. However, the complete optimisation of the logistics activities is highly complicated, and it is therefore often necessary to study smaller sub-problems.

The problem studied in this paper belongs to the category of the optimisation of operations in container terminals. The literature within this field is comprehensive as can be seen in Gharehgozli et al. [11], Meisel et al. [25], and Stahlbock and Voß[35]. Nevertheless, among the great variety of operational problems included in these reviews, there are no conceptually related variants to the PSP. The most common optimisation problems in container terminals comprise the *Berth Allocation Problem* (BAP) and the *Quay Crane Scheduling Problem* (QCSP). For a general survey of these problems, we refer the reader to Bierwirth and Meisel [4]. Although the structure of both problems is quite different from that of the PSP, the mathematical formulation and solution concepts of these problems also involves the scheduling of operations during the vessel planning process at terminals, and have served as inspiration for the development of this work. Kim and Moon [18] present a mathematical formulation for the BAP with continuous berths to determine berthing times and positions of vessels in the ports, and propose a Simulated Annealing heuristic to solve large realistic instances. Sammarra et al. [33] present a mathematical formulation for the QCSP with precedence and non-simultaneously constraints between tasks. The problem can be decomposed into a routing and scheduling problem, and the authors propose a Tabu Search heuristic for solving the routing problem, and a local search heuristic for solving the scheduling sub-problem. The PSP presents a similar problem decomposition, where the routing problem corresponds to the order in which operations are performed at the terminals and for the vessels, and the scheduling problem assigns the starting times of operations. Furthermore, the most employed methodology for solving some variants of the BAP and QCSP are metaheuristics, such as Genetic Algorithms, Simulated Annealing, and Tabu Search [17, 18, 23, 33].

The literature on container terminals is mainly focused on the optimisation of operations from the

terminal point of view. There seems to be no papers dealing with the operational planning of container vessels in terminals seen from the carriers point of view, collaborating with the terminal for a better management of the available resources. The PSP becomes relevant in this case. In our previous work [15], we introduced the PSP, which has been defined in collaboration with representatives of the feeder line industry. The PSP accounts for most of the practical constraints derived from real-life operations. We formulate the problem as a *Mixed Integer Programming* (MIP) model, and propose an ALNS heuristic for solving large instances. The results in [15] highlight the effectiveness of the heuristic, finding high quality solutions and outperforming the mathematical model in almost all cases. In this paper, we study the application of CP techniques for solving the PSP.

Having a closer look at the structure of the problem, we can easily see that the PSP shares many similarities with *machine scheduling problems*, including problems such as *shop scheduling*. These type of problems typically involve the scheduling of jobs to a series of machines, and the most commonly known problems are the job shop scheduling, open shop scheduling and flow shop scheduling problems [27]. However, the latter problems define specific problems in the literature without a wide variety of extra constraints, whereas the PSP covers a more general case of the shop scheduling problems. In particular, the PSP can be identified as a version of the *General Shop Scheduling Problem* (GSP) [6]. Each machine can be seen as a vessel, and each job can be seen as a terminal. Each job consists in a set of operations, which must be processed at specific machines, with individual processing times and without preemption. However, each job does not necessarily need to be processed at all machines. This comes from the fact that a feeder vessel might not need to visit all terminals during a port stay. Moreover, each job can be processed by at most one machine at a time, and each machine can process only one job. Furthermore, there may be some precedence relations between the operations. Up to this point, the PSP can be defined as a machine scheduling problem. Following the terminology from Graham et al. [13], we classify the PSP as $G_m|pred; r_p; d_p; s_{pq}|\sum w_p C_p$. The machine environment corresponds to an m-machine GSP; the job characteristics indicate general precedence relations between operations, release, due and sequence dependent set-up times for operations; and the objective function denotes the weighted sum of the completion times of operations. Nevertheless, the PSP further extends the problem by considering time window constraints for operations, several closing periods at terminals (corresponding to time periods where jobs cannot be executed) and capacity constraints for vessels (or machines). As a generalisation of the GSP, the PSP is $\mathcal{NP}$-hard and difficult to solve in practice for large instances.

In this work, we propose an ALNS framework within the math-heuristic approach to solve large instances of the PSP. The ALNS heuristic has shown to be very efficient in solving several routing and scheduling problems [15, 21, 31, 32]. Additionally, we propose a CP formulation for the PSP, as CP methods have been successfully implemented to solve several variants of scheduling problems [2]. Since the GSP has received little attention in the literature, it is interesting to relate it with other well studied scheduling problems. Grimes et al. [14] present a simple CP model in combination of a weighted-based learning heuristic to solve the open shop scheduling problem. Similarly, Malapert et al. [24] and Gedik et al. [9] present optimal CP approaches for solving the open shop scheduling problem and the parallel machine scheduling problem, respectively, with make-span minimisation. Furthermore, Fleszar and Hindi [8] and

Gokur et al. [12] also propose competitive CP models for the parallel machine scheduling problem with make-span minimisation. In the field of maritime optimisation, Kizilay et al. [20] present MIP and CP models for the integrated optimisation of container terminal operations, and Qin et al. [30] evaluate the solution performance of the constraint and integer programming models for the BAP with additional constraints. Moreover, Qin et al. [29] investigate the joint scheduling problem of container handling operations of a single vessel and propose a hybrid MIP/CP solution strategy to solve the integrated problem.

The combination of local search methods with constraint and mathematical programming models have been successfully applied to several variants of routing and scheduling problems. For solving Vehicle Routing Problems (VRP), Shaw [34] uses a tree-based search with constraint propagation within a local search framework, whereas De Backer et al. [7] introduce a method for using local search techniques within the CP framework. Hojabri et al. [16] investigate the synchronization of vehicles for a variant of the VRP and propose an hybrid scheme that couples the ALNS heuristic with a CP to reconstruct partially destroyed solutions. Furthermore, Watson and Beck [39] and Beck et al. [3] combine a local search algorithm with CP to solve job shop scheduling problems. Gerhards et al. [10] present an ALNS math-heuristic for the multi-mode resource-constrained project scheduling problem, in which a MIP model is used as a repair method within the heuristic framework. Finally, Talbi [36] studies the development of hybrid metaheuristics in the field of optimisation and machine learning, which also includes the combination of metaheuristics with constraint and mathematical programming approaches.

2.2 Test Instances

We consider the *PortLib* instances, a set of benchmark instances introduced in [15]. These instances have been developed to reflect real-life operations for feeder vessels in multi-terminal ports. The instances contain information on the arrival and latest departure time, the initial and maximum cargo capacity, and the priority factor of each vessel. For each operation, the instances include information of the time window, the required service time together with the cargo handling quantity, and the associated vessel and terminal. Furthermore, the instances also contain information about precedence relations between operations, the sailing distance between terminals and the time periods at which the terminals are not operational.

The benchmark suite consists of 300 instances of various sizes, grouped into 15 different terminal-vessel combinations. The instances range from small instances with few operations to large instances representing big congested ports with many feeder vessels visiting multiple terminals. All instances are available online at `Zenodo.com`[1]. Table 1 presents an overview of the main characteristic of the instances.

3 Constraint Programming

Scheduling problems involve the scheduling of operations, tasks or jobs over a time frame while allocating some available resources. This kind of problems can also be composed of additional constraints such as

[1] `https://doi.org/10.5281/zenodo.3820078`

Terminals	2			3				4				5			
Vessels	4	6	8	6	8	10	12	8	10	12	14	10	12	14	16
Operations	12	17	23	26	34	42	51	45	56	68	79	70	84	98	112
Instances	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
Heuristic Run Times (s)	5	5	7	9	20	38	67	46	88	158	247	172	297	471	703
Exact Method Time Limits (s)	50	50	70	90	200	380	670	460	880	1580	2470	1720	2970	4710	7030

Table 1: *Overview of the* PortLib *instances.*

precedence relations between operations, and release and due times for jobs, among many other practical restrictions. Due their nature, scheduling problems seem suitable to be formulated as CP models. CP is a programming paradigm where the user describes the problem to solve in a declarative way, using a modelling language, by means of variables (unknowns) and constraints (relations between variables). Then, the problem is solved by a generic constraint solver. Most of these constraint solvers explore the search space by building a search tree and using constraint propagation to prune the tree. CP belongs to the family of *complete* solution methods, meaning that for optimisation problems, CP can prove the optimality or infeasibility of a problem. Moreover, CP has been successfully applied to solve many scheduling problems. Recently, IBM ILOG developed a model-and-run paradigm to model and solve real-world optimisation problems, with emphasis on planning and scheduling problems, the IBM ILOG CP Optimizer (CPO). The developed tool provided a simple and accessible CP-based paradigm suitable for modelling scheduling problems, which has become the state-of-the art method for many of them [22].

3.1 Notation

We introduce the notation for the PSP that will be used throughout the paper to model the problem using a CP formulation. First, the following sets are used:

- $T = \{0, 1, \ldots, n, n+1\}$: Set of terminals, including dummy terminals (0 and $n+1$) for the entry and exit point of the port, respectively.
- $V = \{0, 1, \ldots, m, m+1\}$: Set of vessels, including dummy vessels.
- O : Set of all operations.
- $\tilde{O} \subset O$: Set of interior operations, excluding dummy operations for terminals and vessels.
- $O_i^v \subset O$: Set of operations to be performed by vessel $v \in V$ at terminal $i \in T$.
- $O^v \subset O$: Set of operations for vessel $v \in V$.
- $O_i \subset O$: Set of operations for terminal $i \in T$.

The parameters required for the CP formulation are:

- δ_{ij} : Sailing distance between terminal $i \in T$ and terminal $j \in T$.

- w_p : Number of containers to be handled at operation $p \in O$. If $w_p < 0$, the operation discharges containers; whereas if $w_p > 0$, the operation loads containers.
- τ_p : Required service time to perform operation $p \in O$.
- λ_{pq} : Binary precedence parameter. Equals to 1 if operation $q \in O$ has to be performed after operation $p \in O$, and 0 otherwise.
- $[\alpha_p, \beta_p]$: Time window for the starting time of operation $p \in O$.
- ϕ_p : Terminal associated to operation $p \in O$.
- ν_p : Vessel associated to operation $p \in O$.
- c_p : Cost coefficient of operation $p \in O$. The precise values for the coefficients are given below.
- $[e_v, l_v]$: Time window for vessel $v \in V$, indicating the arrival and latest departure to and from the port, respectively.
- γ_v : Priority factor for vessel $v \in V$.
- Q_v : Maximum cargo capacity of vessel $v \in V$.
- K_v : Total cargo capacity on-board of vessel $v \in V$ when arriving to the port destined to another port.
- ρ : Penalty weight factor denoting the relative importance for the early departure of vessels from the port.
- F_t : Intensity function for terminal $t \in T$, i.e. the temporal function of the intensity of work within a terminal. The intensity is equal to 0 during the closing times of terminals, otherwise the intensity is set to 100.

Finally, in order to reflect the two primary goals of the objective function on the set of operations, we define the cost coefficients c_p as follows:

$$c_p = \begin{cases} \tau_p \gamma_{\nu_p} & p \in \tilde{O} \\ \rho \gamma_{\nu_p} & p \in O_{n+1} \\ 0 & p \notin \tilde{O} \cup O_{n+1}. \end{cases}$$

For each interior operation $p \in \tilde{O}$, the cost coefficient is given by the required service time and the priority factor of the corresponding vessel. These operations contribute to the minimisation of the weighted starting times of operations. Similarly, for each dummy operation $p \in O_{n+1}$ representing the departure of feeder vessels from the port, the cost coefficient is given by the penalty weight factor and the priority factor of the corresponding vessel. These operations contribute to the minimisation of the departure time of vessels from the port. Finally, we set to 0 the cost coefficients for the remaining operations, as they do not contribute to the objective function.

3.2 Constraint Programming Formulation

In this section, we present the CP formulation for the PSP, which has been modelled and solved using CPO. This formulation presents several advantages, as it reduces the burden of modelling scheduling problems in other mathematical programming tools such as CPLEX. The total number of constraints and decision variables can be largely reduced due to the combinatorial optimisation framework for modelling scheduling problems within CPO, which allows a more flexible definition of modelling concepts such as decision variables or constraints.

We apply interval decision variables to model the non-preemptive operations. An interval decision variable represents an interval of time of a particular activity (or operation) in the schedule whose position in time is unknown [22]. As operations are non-preemptive, the size of the interval variables is fixed and set to the required service time of the operation. Moreover, the domain of these variables is a fixed time interval corresponding to the time window of the starting times of the operations. Additionally, we define dummy interval variables for the discharge operations. These variables are needed for defining the capacity constraints of vessels. This is because CPO does not allow negative values for elementary cumulative expressions, and we model the capacity constraints using modelling concepts from one-to-many-to-one pick-up and delivery problems. We further declare sequence decision variables for terminals and vessels. Here, a sequence decision variable represents a possible temporal ordering of a set of interval decision variables [22]. Each sequence decision variable collects all operations associated to the specific terminal and vessel, respectively. Furthermore, we define cumulative function expressions to represent the temporal cargo evolution of vessels over the time horizon. The on-board cargo destined to another port and the maximum cargo capacity of vessels define the bounds on the cargo evolution for these expressions. Finally, all decision variables are summarised below.

- y_p : Interval variable for the starting time of an operation $p \in O$, defined in $[\alpha_p, \beta_p]$ and of size τ_p.
- y'_p : Dummy interval variable for the starting time of a discharge operation $p \in \{O : w_p < 0\}$, defined in $[e_{\nu_p}, \beta_p]$ and of size in $[\alpha_p - e_{\nu_p}, \beta_p - e_{\nu_p}]$.
- S_t^T : Sequence variable for terminal $t \in T$ over $\{y_p \mid p \in O_t\}$.
- S_v^V : Sequence variable for vessel $v \in V$ over $\{y_p \mid p \in O^v\}$.
- C_v : Cumulative function expression for vessel $v \in V$ for the cargo load, defined in $[K_v, Q_v]$.

Once the notation and decision variables have been defined, the CP model for the PSP can be expressed as follows.

Objective:

$$\text{minimise} \sum_{p \in \bar{O} \cup O_{n+1}} c_p \ \texttt{startOf}(y_p) \tag{1a}$$

Constraints:

$$\texttt{noOverlap}(S_t^T) \qquad t \in T \qquad \text{(1b)}$$

$$\texttt{noOverlap}(S_v^V, \delta_{ij}) \qquad v \in V \qquad \text{(1c)}$$

$$\texttt{endBeforeStart}(y_p, y_q) \qquad p, q \in \{O : \lambda_{pq} = 1\} \qquad \text{(1d)}$$

$$\texttt{forbidExtent}(y_p,\ F_t) \qquad t \in T, p \in O_t \qquad \text{(1e)}$$

$$C_v = \sum_{\substack{p \in O^v: \\ w_p > 0}} \texttt{stepAtStart}(y_p,\ w_p) + \sum_{\substack{p \in O^v: \\ w_p < 0}} \texttt{pulse}(y'_p, w_p) \qquad v \in V \qquad \text{(1f)}$$

$$\texttt{endAtStart}(y'_p, y_p) \qquad p \in \{O : w_p < 0\} \qquad \text{(1g)}$$

The objective function (1a) minimises the weighted sum of the starting times of operations and the sum of the departure times for the vessels. This is reflected in the objective function by the sum over the set of interior operations $\tilde{O}$, and by the sum over the set of dummy operations O_{n+1} representing the departure of feeder vessels from the port. The cost coefficients c_p collect the corresponding weight factor for each operation, and the expression $\texttt{startOf}(y_p)$ over the interval decision variable y_p returns the starting time of an operation $p \in \tilde{O} \cup O_{n+1}$.

The disjunctive resources constraints are defined in Constraints (1b) and (1c). These constraints are expressed by the `noOverlap` constraints on the sequence variables, which state that the sequence defines a chain of non-overlapping interval variables. The agreements between the terminals and the carrier state that a terminal $t \in T$ cannot serve more than a single vessel at a time. This is ensured in Constraints (1b). Furthermore, a vessel $v \in V$ cannot be simultaneously at two different terminals. Constraints (1c) define the non-overlapping constraints on the sequences defined by the vessels' operations. Moreover, these non-overlapping constraints include the transition distance matrix δ_{ij} to model the sailing distance between terminals.

Constraints (1d) define the precedence relations between operations. This is modelled by the temporal constraint $\texttt{endBeforeStart}(y_p, y_q)$, which ensures that operation $p \in O$ must be completed before starting operation $q \in O$. Moreover, Constraints (1e) forbid operations to overlap with time periods in which a terminal $t \in T$ is not operational. This is modelled by the constraint $\texttt{forbidExtent}(y_p,\ F_t)$, which does not allow an interval decision variable for an operation $p \in O_t$ to extend over a time period where the intensity function F_t is 0.

Lastly, Constraints (1f) and (1g) define the capacity constraints of the vessels. The temporal cargo evolution for vessel $v \in V$ during the port stay is defined by Constraints (1f). In this expression, we see that the cargo evolution depends on the type of operation (loading or discharging containers) at the terminal. The first term accounts for the loading operations ($w_p > 0$), which increases the cargo level on the vessel at the starting time of the loading operation. These operations are modelled with `stepAtStart` functions, as the loaded cargo corresponds to cargo to be delivered to other ports in the region, and it does not reduce the cargo level of the vessel until the vessel leaves the port. The second term accounts for the discharge operations ($w_p < 0$), which decreases the cargo level on the vessel at the beginning of the

discharge operation. These operations are modelled with `pulse` functions on the dummy interval variables y'. As the discharge operations handle containers that are already on-board the vessel when arriving to the port, the `pulse` function increases the cargo level at the arrival time of the vessel. At the beginning of the discharge operation, the vessel starts discharging containers, and therefore, the cargo level decreases. Figure 2 shows the elementary cumulative expressions for loading or discharging containers for a given interior operation $p \in \tilde{O}$.

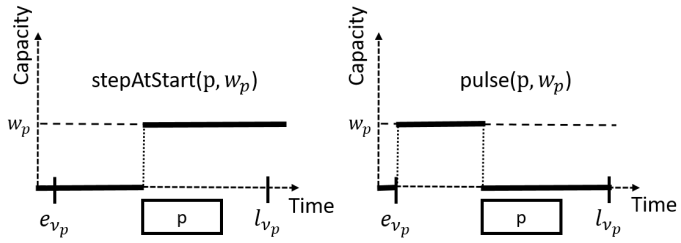

Figure 2: *Elementary cumulative expressions for loading containers (left) and discharging containers (right) for a given interior operation $p \in \tilde{O}$. At the beginning of operation p, the* ***`stepAtStart`*** *function increases the number of containers on-board, whereas the* ***`pulse`*** *function decreases the cargo level.*

The cargo capacity also includes the containers on-board the vessel when arriving to the port, but not destined to be delivered at the port. Additionally, the maximum cargo capacity of the vessel should not be exceeded at any time, and this is reflected by the bounds of the cumulative function expression. Furthermore, Constraints (1g) define the channelling constraints between the discharge interval variables. These constraints ensure that as soon as the discharge operation $p \in \{O : w_p < 0\}$ begins, the pulse for the dummy discharge interval variable must end. This is modelled by the temporal constraint `endAtStart(`y'_p`,`y_p`)`. One could also model the loading operations following the same reasoning, i.e. defining dummy interval variables for these operations. However, initial testing has shown that adding extra redundant interval variables for the loading operations does not provide any apparent benefit. The interval variables y are sufficient for keeping track of the loading operations. Nevertheless, the dummy interval variables for the discharge operations are necessary for modelling the capacity constraints, and they need to be included. These variables do not add unnecessary overhead to the model, as the time assignment of one channel variable implies the time assignment of the other. Finally, for a better understanding of the cargo evolution of a vessel, Figure 3 depicts an example of the temporal capacity of the vessel during a port stay.

3.3 Computational Performance

In the following section, we study the performance of the proposed CP model on the *PortLib* instances. The CP model has been implemented in Java, and solved using the IBM ILOG CP Optimizer version 12.9 on a Huawei XH620 V3 computer with 2.6 GHz Intel Xeon Processor 2660v3. The complete results can be found available online at `Zenodo.com`[2].

[2] `https://doi.org/10.5281/zenodo.3820078`

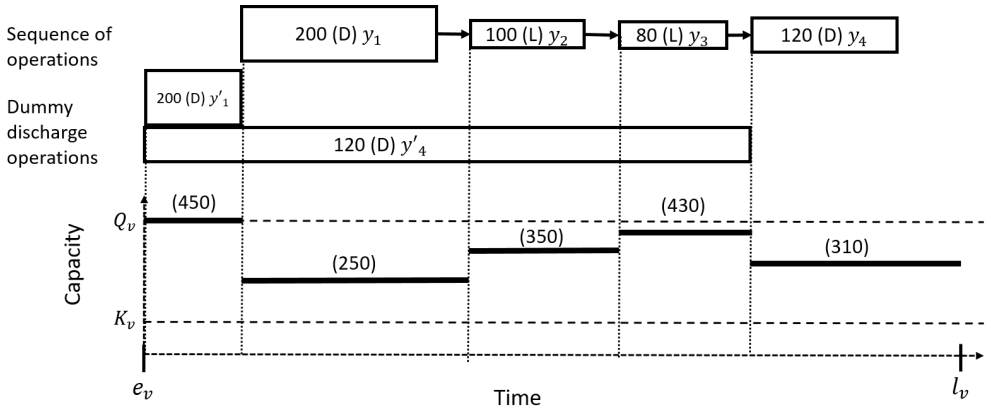

Figure 3: *Example of the cargo evolution of a vessel during a port stay. The vessel performs two loading and two discharge operations. The operations are represented by rectangles, whose length is given by the service time. The type of operation (D: Discharge, L: Loading) and the number of containers to handle are written within the rectangle. The dummy discharge operations are also represented by rectangles. However, its length corresponds to the total on-board time of the containers to discharge. The vessel has a maximum capacity of* $Q_v = 450$ *containers, and arrives to the port with* $K_v = 130$ *containers destined to another port.*

For comparison reasons, we decide to compare the results against the MIP model presented in [15]. The CP model and the MIP model are solved using commercial constraint and mathematical programming optimisation tools: CPO and CPLEX solvers, respectively. Both tools use *complete* solution methods, and it seems to provide a fair comparison between the models. Furthermore, we set a maximum time limit in all experiments as specified in Table 1. Finally, we report the deviation of the solutions with respect to the best-known solution as comparison measure between the methods.

3.3.1 Search Parameterisation

A CP solver such as CPO uses constraint propagation techniques and search heuristics for solving optimisation problems. These techniques are used to guide the search for new solutions. However, the search can be influenced by many search parameters and search phases. We consider the default parameter setting of CPO. Nevertheless, we present different ways to parametrise the automatic search, derived from the main characteristics of the problem and as presented in [22]. These considerations have shown to improve the performance of the default configuration of CPO, obtaining overall better solutions.

Alternative Search Type: During the execution of CPO, we use the default search, i.e. the *restart* search. This search type controls the search for new solutions through progressive restarts. Although the *restart* search is the most effective for most combinatorial problems, the search may stagnate and not find feasible solutions for strongly constrained problems. If the *restart* search fails to find an initial feasible solution, we use *multi-point* search to generate an initial solution. The *multi-point* search combines a set of generated solutions for producing better solutions. After a feasible solution is obtained, the solver continues the search using the *restart* search with the previously found solution as starting point.

Search Phases (2Ph): The domain of decision variables can be reduced during the search, since CPO can assign values to some variables and propagate constraints to filter the domain of the unassigned variables. If the current assignment is not successful, the domain of the decision variables can be restored by backtracking the search. The order in which the decision variables are assigned is important. Assigning certain variables before others can result in faster convergence or in obtaining better solutions. Therefore, we consider the instantiation of the decision variables using two search phases, where the decision variables in the first phase are instantiated before the decision variable in the second phase. The first phase accounts for all interior operations $p \in \tilde{O}$, whereas the second phase accounts for the remaining dummy operations $p \in O \setminus \tilde{O}$.

Generally, the set of dummy operations for the vessels' departure from the port have a higher cost coefficient, as earlier departures of vessels from the port is a priority for the shipping company. However, these operations are given as a result of scheduling all the corresponding operations for the vessel within the port, as vessels depart from the port as soon as all operations have been performed. Taking this into account, we separate these operations into different phases, in order to prioritise the search for the interior operations.

No Overlap Inference Level (nOE): One of the main features of the problem comes with the definition of the disjunctive resources constraints. This assumption is important, as it limits the maximum resource levels for terminals and vessels to process up to one operation at a time. This enforces that operations must not be carried out simultaneously on the resource. This is modelled using the non-overlapping global constraints. In order to achieve a more thorough domain reduction on the decision variables, we set the inference level of these constraints to *extended* values.

3.3.2 CPO Performance and Comparison with CPLEX

In the following section, we study the performance of CPO. First, we test CPO with the default parameter setting, and we compare the performance with the different combinations of the aforementioned search parameterisation. For all experiments, we consider the alternative search type as presented presented in Section 3.3.1, as feasible initial solutions for all instances can be obtained almost immediately.

In order to compare the performance of the methods on the set of all instances, we report the average deviation over the best-known solution achieved by the methods. For a given method d, we calculate the average deviation as $\frac{1}{n}\sum_{i=1}^{n} 100 \cdot (\frac{\hat{z}_i^d}{z_i^*} - 1)$, where n is the total number of instances, $\hat{z}_i^d$ the objective value for instance i achieved by method d, and z_i^* is the best-known objective value for instance i. Furthermore, for a more complete comparison, we also include the CPLEX results for MIP model from [15]. All instances have been run with a maximum execution time as shown in Table 1, and the results can be found in Table 2. The row **Settings** indicates the activated search parameterisation: *2Ph* states two search phases and *nOE* states an extended inference level on the non-overlapping constraints. The row **Average Deviation** reports the average deviation of all instances with respect to the best-known solution achieved by the methods. Finally, the row #**Optimal** indicates the number of instances where the method proves optimality, the row #**Best** indicates the number of instances where the method finds the best-known solution, and the

row #**Single** indicates the number of instances where the method is the only one to find the best-known solution.

Method	CPLEX	CPO			
Setting	Default	Default	2Ph	nOE	2Ph+nOE
Average Deviation (%)	1.35	0.61	0.68	0.47	0.46
#Optimal	109	121	122	122	121
#Best	141	171	169	184	192
#Single	20	23	19	39	39

Table 2: *Summary of the results for the solution methods with different search parameterisation.*

We see from the results that, in general, CPO performs better than CPLEX. For all different search parameterisations, CPO reports average deviations smaller than 0.7%, finding the best-known solution for more than 170 instances and around 120 optimal solutions. On the other hand, CPLEX reports in comparison a high average deviations of 1.35%, where the method is only able to find the best solution for 141 of the 300 instances and finds 109 optimal solutions. Additionally, Figure 4 shows the average deviation of the different solution methods for each of the terminal-vessel combinations. The results show low average deviations for all configurations of CPO settings, reporting values always below 2%. In contrast, the average deviation for CPLEX is consistently higher for all cases, being higher than 2% from instances with more than 55 operations. Furthermore, we see that the average deviation of CPO can be improved by activating the two search phases and increasing the categorical inference level of the non-overlapping constraints. In general, this configuration (CPO 2Ph+nOE) reports the overall best performance for all terminal-vessel combinations. As seen in Table 2, the average deviation of the default configuration can be reduced from 0.61% to 0.46%, and the number of instances where we find the best-known solution is increased from 171 to 192. It is further interesting to see that, when activating the two search phases (Configuration CPO 2Ph), the number of best-known solutions slightly decreases, and the average deviation increases to 0.68% with respect to the default configuration. However, combining this configuration with an extended inference level of the non-overlapping constraint (i.e.CPO 2Ph+nOE), the CPO performance is improved and a lower average deviation of 0.46% is reported.

To have a clearer picture of the performance of CPO against CPLEX, we compare in Figure 5 the ratio between the solutions found by CPLEX and by CPO with default configuration for all *PortLib* instances. The graph groups the instances by the same number of terminals, and in each group, the instances are sorted by increasing number of vessels. We see from the graph that both methods find the same solution for smaller instances, where the ratio between the solutions is always 1. For all instances with 2 terminals, and for instances with 3 terminals and few vessels, both methods converge and proved optimality within the given time frame. From this point, we see how the solution-quality of the found solutions by both methods begins to differ. Some optimal solutions can still be found for some instances with larger number of operations, though only feasible solutions are achieved for the largest instances within the time limit. In general, CPO outperforms CPLEX in terms of solution-quality. As represented in the graph, CPO tends to find better solutions than CPLEX, with a clear predominance of ratios greater than 1, where

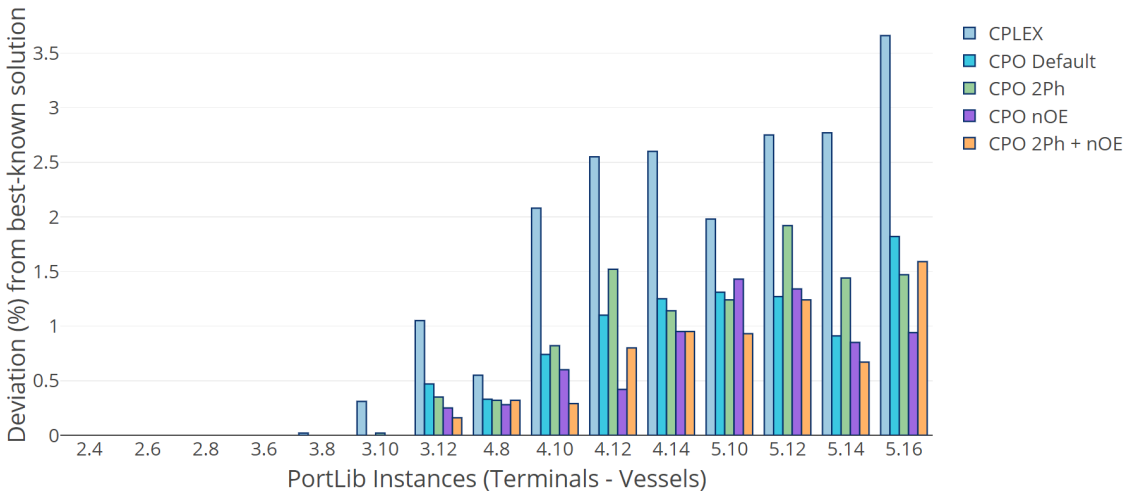

Figure 4: *Average deviation of the different solution methods with respect to the best-known solution. The instances are grouped by the number of terminals and number of vessels.*

some differences reach up to 8% for some of the larger instances. Nevertheless, CPLEX still returns better solutions for some few cases. From the 300 instances, CPO finds better solutions for 141 instances, whereas CPLEX only finds better solutions for 39 instances. For the remaining 120 instances, CPO and CPLEX found the same solution.

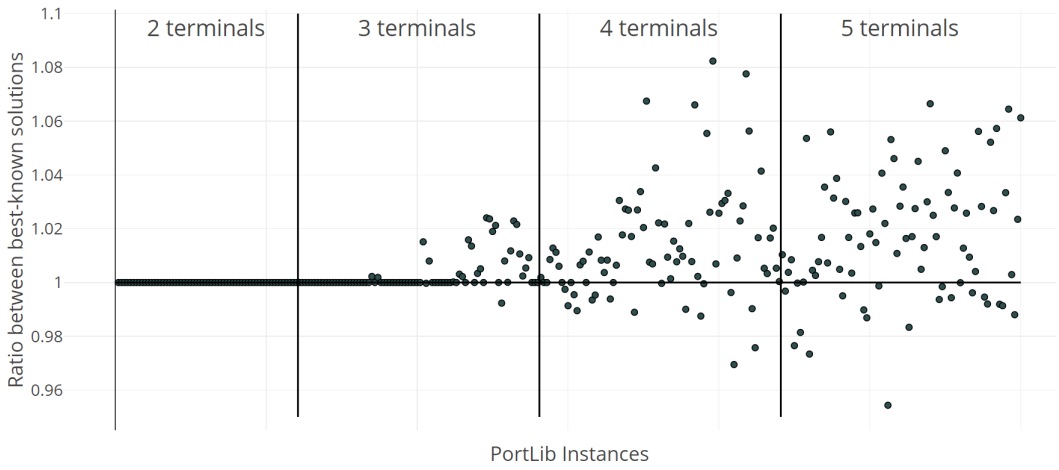

Figure 5: *Ratio between the solutions found by CPLEX and CPO. A value above 1 means that CPO finds a better solution than CPLEX within the time limit. The instances have been sorted by increasing number of terminals and vessels.*

To further study the performance of the methods, we show in Figure 6 the evolution of the current best solution of both methods over the time frame for instance `PSP.5.12.16`, the instance with the lowest ratio from Figure 5. For this instance, CPLEX was able to find a solution that is 4.57% better than the solution returned by CPO. As shown in the graph, CPO finds good quality solutions quite quickly. During

the early stages of the search, CPO continuously improves the current solution, and later on, the search stagnates without further improving the best found solution. On the other hand, the evolution of the current solution for CPLEX is more controlled. We see that after 300 seconds, the current best solution for CPLEX is about 3.5% worse than the solution obtained from CPO. Throughout the search, CPLEX maintains a lower quality solution compared to the one obtained by CPO at the beginning. However, CPLEX performance remains consistently steady, slowly improving the current solution and finding a better solution only towards the end of the time limit. This is a clear example of the difference between the search engines of the two solvers.

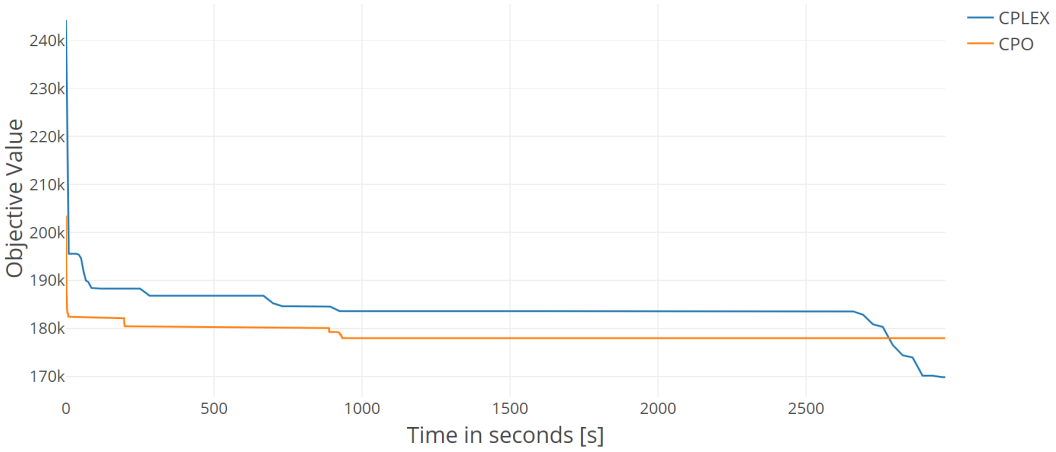

Figure 6: *Evolution of the current best solution for CPO and CPLEX for instance* `PSP.5.12.16`.

The rapid increase in solution-quality during the first stages of the search reported by CPO is observed in basically all *PortLib* instances, with some exceptions. A CP engine finds new solutions by assigning values to the decision variables and using constraint propagation to reduce the domains on the remaining variables. Moreover, this engine proves optimality when no better solution than the current solution can be found. On the other hand, a mathematical programming engine uses techniques from branch and bound and linear relaxations to guide the search. Therefore, a CP engine is very effective in quickly finding good solutions, though it will have difficulties in later stages if the solution space is large and the search is trapped in areas where the optimality of the current solution cannot be proved. Although CPLEX may return better solutions for some instances, this dominance is restricted to long-term performance of the solver. In order to test this, we depict in Figure 7 the time evolution of the average deviation over the best-known solution for all *PortLib* instances when we increase the time limit for the different solvers to 7,200 seconds. The figure shows how the CP solver is able to quickly explore the solution space in the early stages of the search, improving very efficiently the current solution. After that, the CP solver maintains a regular performance with minor improvements until the termination criteria is met. We further see how the search parameterisation improves the overall performance of the CP solver. On the other hand, the figure shows how the curve evolution for CPLEX is considerably worse than that obtained by CPO throughout the entire execution period. The mathematical programming engine constantly improves the current solution,

although we can see that the search is trapped in a less promising area of the solution space, from which it escapes only after long execution times. The rapid convergence towards good solutions for CPO suggests that a combination of this method with local search heuristics may prompt to improving performance.

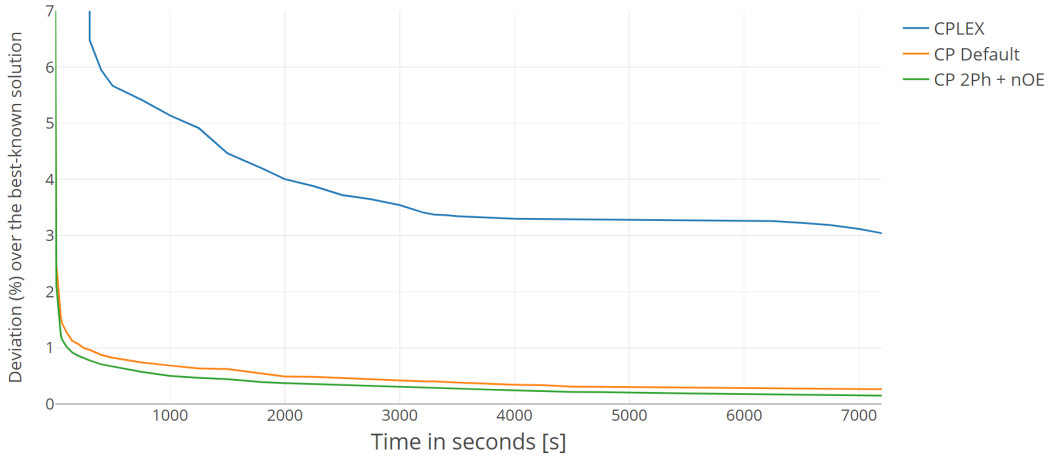

Figure 7: *Evolution of the average deviation over the best-known solution for the different solvers (Average of all* PortLib *instances).*

4 The Adaptive Large Neighbourhood Search Math-heuristic

In the following section, we present an ALNS math-heuristic for solving the PSP. The proposed approach is based on the integration of a local search method in collaboration with an exact method. Hence, we classify this approach as a math-heuristic method [5]. As exact method, we consider CPO for solving the CP model presented in Section 3.2. As local search method, we consider an adapted version of the ALNS heuristic from our previous work in [15]. A more detailed presentation of the local search method and the general framework of the math-heuristic is given below.

4.1 The Local Search Framework

The *Large Neighbourhood Search* (LNS) heuristic was first introduced by Shaw [34]. This heuristic follows a ruin-and-recreate principle, in which an initial solution is continuously altered by means of destroy and repair methods. Destroy methods randomly remove a part of the current solution, whereas repair methods rebuild the previously destroyed partial solution into a complete solution using greedy methods. Later on, Ropke and Pisinger [32] introduce the *Adaptive Large Neighbourhood Search* (ALNS) heuristic, which is an extension of the LNS heuristic. Within this framework, the destroy and repair methods are chosen statistically based on the previously achieved performance during the search.

In this paper we use an adapted version of the ALNS heuristic from [15]. One of the most important key factors for the development of the heuristic is the separation between the order and the time assignment

of operations. A solution of the problem is represented by an acylic orientation of the disjunctive graph, and the starting time of operations can be optimally assigned as part of the solution evaluation through Dynamic Programming. For more details on the problem decomposition, we refer to the seminal paper [15].

The former heuristic presents an adaptive framework for the selection of a pair of destroy and repair methods. Although this promotes the selection of methods that work well together, the number of combinations can grow rapidly, especially if destroy methods are defined for many different sizes, and repair methods can follow several strategies. To prevent loss of information during the search with a large number of combinations, we reduce the number of methods and consider separate weights for the destroy and repair methods. This is done for example in the original ALNS framework presented in Ropke and Pisinger [32].

Let Ω^+ and Ω^- denote the set of destroy and repair methods, respectively. At each iteration, the heuristic selects a destroy method $d \in \Omega^+$ and a repair method $r \in \Omega^-$ to modify the current solution based on the current probability of the methods. The methods are chosen statistically using the *roulette wheel selection principle*. All weights are initialised with equal probability, and are updated iteratively according to the quality of the obtained solution. At each iteration, if the heuristic returns an improving solution, the weights of the selected methods are increased by a factor of π^+; if the heuristic returns a deteriorated solution, the weights of the selected methods are decreased by a factor of $(1 - \pi^-)$; whereas if the heuristic returns the same current solution, the weights are not updated. As the weights represent probabilities, the remaining weights also need to be accordingly adjusted.

To control the exploration of the solution space of the heuristic, we use the classic temperature acceptance criteria from Simulated Annealing, as first introduced by Kirkpatrick et al. [19], and also presented in Ropke and Pisinger [32]. The heuristic makes use of a temperature parameter T to control the acceptance probability of the new generated solution at each iteration. Let s be the current solution, and let s' be the resulting solution after applying the destroy and repair method at a given iteration. The new generated solution will be accepted with probability $e^{\frac{f(s)-f(s')}{T}}$, where $f(s)$ denotes the objective value of solution s. The temperature is initialised as a factor T_{st} of the objective value of the initial solution, and decreases exponentially throughout the search towards a factor T_f of the objective value of the initial solution, reducing the margin of acceptance in the final stages. Furthermore, we consider time as stopping criterion, and we control the temperature parameter using the elapsed time. Finally, we endow the heuristic with a restoring feature, returning to the best-known solution after a certain number of iterations without improvement, and increasing the temperature to half of the initial temperature.

At each iteration, the ALNS heuristic destroys part of the current solution. The parameter b determines the number of operations to remove, and is randomly chosen from the interval $[1, B]$, where B is the maximum allowable number of operations to remove. This is done to add some diversification into the search, and to reduce the large number of destroy methods. Furthermore, we denote $\hat{O}$ as the set of removed operations. We define two destroy methods:

- **Random Removal:** Remove b random operations from the current solution.

- **Predecessors and Successors Removal:** Select a random operation p to remove from the current solution. Remove as well the direct predecessors and successors operations in the vessel's and terminal's route. Repeat until b operations have been removed.

Then, the ALNS heuristic repairs the previously destroyed solution using the greedy insertion method, i.e. for a given operation $p \in \hat{O}$, the operation is inserted in the current solution at the position where the resulting objective value increases the least. We consider different sorting strategies for the removed operations, and therefore, we define four repair methods. The operations are inserted in the current solution according to the sorting strategies: (1) No sorting, i.e. operations are sorted as they are removed; (2) sorting in decreasing duration of the service time; (3) sorting in decreasing size of the operational time window; and (4) sorting in decreasing time of the latest starting time of the operational time window.
Finally, as infeasible solutions can be obtained during the repair of the current partial solution, the heuristic allows infeasibility by incurring a high penalty cost during the solution evaluation.

4.2 The General Math-heuristic Framework

The local search framework allows the heuristic to explore areas of the solution space through a ruin-and-recreate procedure. Although this search is effective, it heavily depends on the definition of the greedy methods and the acceptance probability criteria to exploit promising areas of the search space. To compensate for the randomness within the heuristic, and in order to move from local optimums, many iterations may be required. On the other hand, a CP solver uses constraint propagation to filter the solution space. The combination of the two solution methods aims to take advantage of the search engines of each method, and thus to improve the overall performance of the heuristic. Next, we present the general math-heuristic framework of the solution approach, where the ALNS heuristic interoperates with the CP solver during the search.

We embed the math-heuristic within a particle swarm framework [28]. In this framework, a population of candidate solutions explore independently the solution space. Although it is advisable to always start the search with the best-known solution, we risk of not being able to escape from the local optimum. Furthermore, many local search heuristics are sensitive to the initial search point. Therefore, in some cases, providing a population of candidate solutions may result in more successful searches, as the heuristic can explore different areas of the solution space. In our framework, we consider a population of two initial candidate solutions to initialise the search of the ALNS math-heuristic:

- The first candidate solution is obtained using CPO with maximum execution time of t_0 seconds.
- The second candidate solution is obtained using simple improving heuristics. We use the construction procedure based on several sorting rules for the insertion of operations. For more information, Appendix A describes the algorithm procedure.

The heuristic begins to explore the solution space from the two starting points following the particle swarm framework. This promotes diversification within the heuristic search. The particle swarm framework shares similarities with the general framework of *go with the winners* algorithms [1]. During the execution of

the heuristic, the particles (or candidate solutions) work independently, only communicating and updating the current solution of each particle if a new best solution is found. When this happens, the search continues using the CP engine. The CP solver is then initialised with the best-known solution as the starting point, and performs a continuous search until a certain number μ of failures without improving the current best solution is obtained. Remark that this improving step is also performed on the initial population of candidate solutions. Then, the heuristic search continues until a new best solution is obtained. The heuristic search is very volatile, though it allows to quickly explore different areas of the solution space. When the heuristic finds a new best solution, the CP solver can be used to intensify the search using constraint propagation to reduce the domains of the decision variables.

Algorithm 1 provides a high-level pseudo-code for the general framework of the math-heuristic. At the beginning of the math-heuristic, we initialise the weight parameters to have equal probability. We denote $\rho^- \in \mathbb{R}^{|\Omega^-|}$ and $\rho^+ \in \mathbb{R}^{|\Omega^+|}$ the weight vectors for the destroy and repair methods, respectively. Next, in lines 2-4, we generate and store the initial set of candidate solutions: s^{IH} is the initial solution obtained using simple improving heuristics, whereas s^{CP} is the solution obtained using CPO. The current best solution is denoted by s^*, and the function $BestOf()$ retrieves and stores the best initial solution, as seen in line 4. Moreover, the heuristic initialises the temperature as a factor of the objective function of the initial solution in line 5.

Algorithm 1: Pseudo-Code for the ALNS Math-heuristic.

```
1  Set initial weights ρ⁻ and ρ⁺ with equal probability, and initialise the iteration parameter it;
2  s^IH ← InitialSolutionLocalSearch() ;
3  s^CP ← InitialSolutionConstraintProgramming(t_0) ;
4  s* ← BestOf(s^IH, s^CP) ;
5  Initialise Temperature: T = T_st · f(s*) ;
6  while stopping criteria is NOT met do
7      s ← SelectCandidateSolution(s^IH, s^CP, it) based on current iteration it ;
8      Select a destroy method d() ∈ Ω⁻ using ρ⁻ ;
9      Select a repair method r() ∈ Ω⁺ using ρ⁺ ;
10     s' ← r(d(s)) ;
11     if accept(s', s) then
12         s ← s' ;
13     Update ρ⁻ and ρ⁺ based on acceptance criteria ;
14     if f(s) < f(s*) then
15         ŝ ← s;
16         while f(ŝ) ≠ f(s*) do
17             ŝ ← s*;
18             s* ← ConstraintProgramming(ŝ, μ) ;
19         (s^IH, s^CP) ← s* ;
20     Increase iteration parameter it ;
21     Update temperature parameter T based on elapsed time ;
22 return s* ;
```

The subsequent while loop of lines 6-21 describes the ruin-and-recreate procedure under a simulated annealing scheme to search for improving solutions. First, at the beginning of each iteration, the heuristic selects the current solution from the population of candidate solutions, as seen in line 7. We use a simple selection policy, where the heuristic alternates between the population of candidate solutions at each iteration. Next, we generate a new solution by selecting a destroy and a repair method using a roulette wheel selection. This is done in lines 8-10. In here, $d(s)$ represents the partial solution achieved after applying the destroy method $d() \in \Omega^-$ to the solution s, and equivalently $r(s)$ is the complete solution from applying the repair method $r() \in \Omega^+$ to s. The acceptance of the new generated solution is carried out in lines 11-12. The weights of the methods are accordingly updated based on the acceptance criteria, as seen in line 13. Every time a new best-known solution is found, the heuristic switches the search engine to use CP. This is summarised in lines 14-19. In the following step, we intensify the search by calling the CP solver with the new current best-known solution as initial starting point. This process is repeated while a new improving solution is obtained, as seen in lines 16-18. The CP search continues until a certain number μ of failures are obtained since the last improving solution. At the end of the loop, in line 19, we store the new best solution and set the population of candidate solutions to the new best found solution. Finally, we increase the iteration parameter and update the temperature parameter based on the elapsed time, in lines 20 and 21, respectively.

4.3 Computational Performance

In the following section, we study the performance of the proposed math-heuristic on the *PortLib* instances. The math-heuristic approach has been implemented in Java, and the CP model has been solved using the IBM ILOG CP Optimizer version 12.9. Both methods have been run on a Huawei XH620 V3 computer with 2.6 GHz Intel Xeon Processor 2660v3. The complete results can be found available online at `Zenodo.com`[3].

The math-heuristic uses an adaptive search framework during the heuristic search, and the parameter setting of the math-heuristic has been set to the same values as documented in [15]. We set the maximum initial execution time for CP to $t_0 = 60$ seconds, since it corresponds to the beginning of the stagnation of the average deviation for CP as seen in Figure 7. Moreover, we set the maximum number of failures within the CP engine to $\mu = 75,000$. Furthermore, we also consider the running times of the math-heuristic to be the same as in the seminal paper [15]. The exact running times of the *PortLib* instances can be seen in Table 1. In this section, we study the performance of the ALNS math-heuristic and the efficiency of incorporating CP techniques during the search. For this purpose, we compare the solutions of the ALNS math-heuristic to the solutions obtained from other methods. Due to the stochastic behaviour of the adaptive search framework, we run the math-heuristic 10 times on each of the *PortLib* instances, in order to reduce the variance in the results. For a more complete comparison, we also include the results of the ALNS heuristic and CPLEX from [15]. Finally, we report throughout the experiments the average deviation of the solutions with respect to the best-known solution as comparison measure between the methods. For each stochastic method, we consider two different key performance indicators: the average

[3] `https://doi.org/10.5281/zenodo.3820078`

and the minimum deviation over the best-known solution, respectively. Let μ_i^d and $\hat{z}_i^d$ be the average objective value and the best objective value, respectively, for instance i achieved by method d. Moreover, let z_i^* be the best-known objective value for instance i, and n the total number of instances. The average deviation compares the average performance of the method with respect to the best-known solution, i.e. it is calculated as $\frac{1}{n}\sum_{i=1}^{n} 100 \cdot (\frac{\mu_i^d}{z_i^*} - 1)$; whereas the minimum deviation compares the best solution found by the method against the best-known solution, which is calculated as $\frac{1}{n}\sum_{i=1}^{n} 100 \cdot (\frac{\hat{z}_i^d}{z_i^*} - 1)$.

	Solution Method			
	Math-heuristic	**ALNS**	**CP/CPO**	**MIP/CPLEX**
Average Deviation (%)	0.53	0.95	0.90	1.80
Minimum Deviation (%)	0.09	0.29		

Table 3: *Comparison of the average and minimum deviations for the different solution methods on the* PortLib *instances. Remark that, for the exact methods (CP and CPLEX), the average and the minimum deviation are equivalent.*

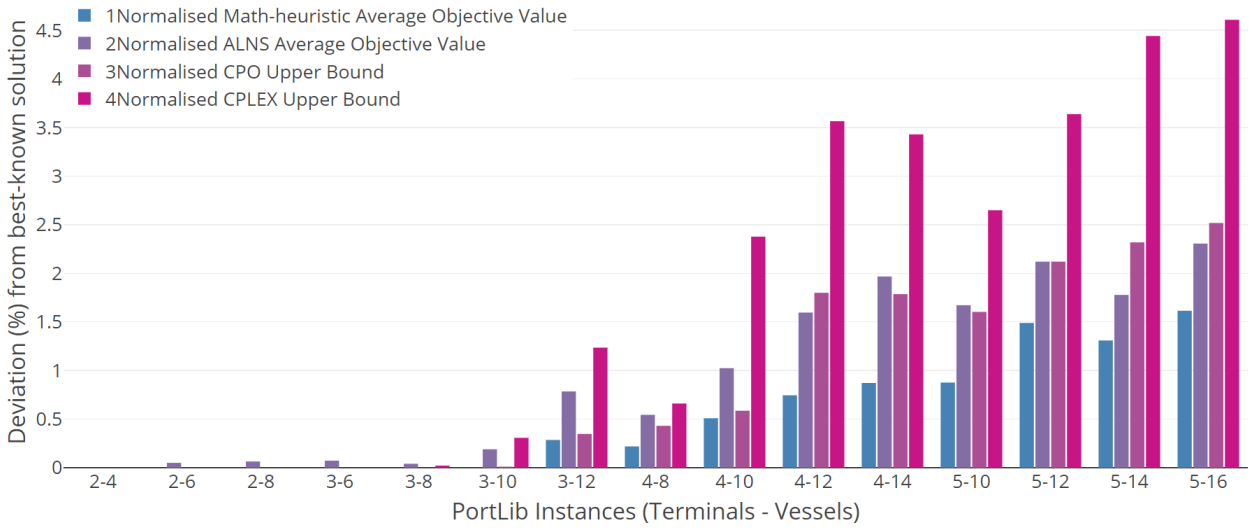

Figure 8: *Average deviation of the different methods. The graph plots the average deviation for each of the terminal-vessel combinations of the* PortLib *instances.*

Table 3 summarises the comparison of the average and minimum deviation of the different solution methods. Moreover, Figure 8 depicts the aggregated average deviation for each terminal-vessel combinations of the *PortLib* instances.

In Section 3.2, we already saw how the complexity of the problem increases rapidly. For large instances with more than 50 operations, CPO and CPLEX reached the maximum time limit without being able to prove optimality. Nevertheless, the results showed that the CPO performance was considerable better than CPLEX. In this section, we include the results from the heuristic methods, and we see that the average deviation for CPO and CPLEX raises to 0.9% and 1.80%, respectively. Additionally, we see in Figure 8 that the average deviation for CPLEX is always higher than the rest of the solution methods.

From our previous work in [15], we report promising results for the ALNS heuristic on the *PortLib* instances. The ALNS heuristic clearly outperforms CPLEX in nearly all cases. However, the superiority of the ALNS heuristic against CPO is not so clear. In general, the ALNS heuristic returns better quality solutions in most cases with considerable shorter execution times than CPO, and the minimum deviation of the ALNS heuristic is consistently lower. However, the heuristic is highly stochastic, and the average deviation is not as stable, reporting a higher average deviation of 0.95% in comparison to value of 0.9% reported by CPO. Moreover, the ALNS heuristic and CPO report similar performance for each terminal-vessel combination, as seen in Figure 8. This difference between the average and minimum deviations of the methods provides the first insights of the possible benefits when combining both methods. We should further note that the maximum execution time of CPO has been set to 10 times the run times of the heuristic. Although this can result in an unbiased comparison, we see that, even with longer running times, CPO is clearly outperformed by the math-heuristic. The results show that the math-heuristic approach reports the lowest average deviation of 0.53%, and in average, the math-heuristic is consistently better than all of the other methods, reporting always lower average deviations for all terminal-vessel combinations, as seen in Figure 8.

The incorporation of CP techniques within the heuristic search can lead to overall improvements. The benefits of using a CP engine during the heuristic search are significant. As we can be seen from the results in Table 3, the math-heuristic approach reports a noteworthy lower average and minimum deviations, 0.53% and 0.09%, respectively, outperforming the ALNS heuristic, CPO and CPLEX. The solutions obtained by the math-heuristic are also much more stable. The average deviation is almost 50% lower than that achieved by the ALNS heuristic. Furthermore, the quality of the solutions returned by the math-heuristic are quite high, being quite close in all cases to the best-known solutions.

To further study the performance of the math-heuristic, we compare the results for different configurations of the math-heuristic on the *PortLib* instances. The interest of this comparison is twofold: (1) to study if the particle swarm framework helps to improve performance, and (2) to study if using a CP engine within the heuristic search can also improve performance. For this purpose, we consider four different configurations of the math-heuristic:

A. ALNS math-heuristic with particle swarm framework and CP engine within the heuristic search.

B. ALNS math-heuristic with particle swarm framework without CP engine within the heuristic search.

C. ALNS math-heuristic with single particle framework and CP engine within the heuristic search.

D. ALNS heuristic with single particle framework without CP engine within the heuristic search.

The comparison results are reported in Table 4. Moreover, in Figure 9, the instances have been grouped by number of terminals and by number of vessels. The graph depicts the results for the average deviation of the different heuristic configurations.

We see from the results in Table 4 that the heuristic configuration (Configuration D), corresponding to the solution approach that does not use any CP technique, presents the highest average and minimum

	Configuration			
	Conf. A	**Conf.B**	**Conf. C**	**Conf. D**
Average Deviation (%)	0.6	0.75	0.81	1.14
Minimum Deviation (%)	0.16	0.28	0.18	0.34
Particle Swarm	×	×		
CP Engine	×		×	

Table 4: *Comparison of the average and minimum deviations for the math-heuristic approach with different configurations on the* PortLib *instances. The configurations have been sorted by increasing average deviation.*

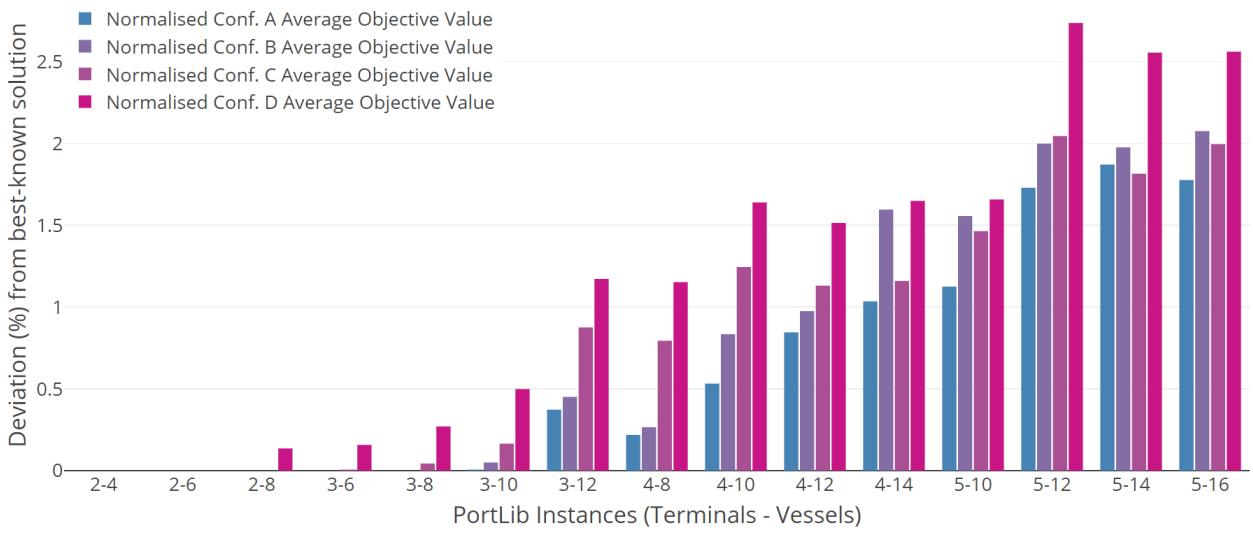

Figure 9: *Average deviation of the different heuristic configurations. The graph plots the average deviation for each of the terminal-vessel combinations of the* PortLib *instances.*

deviation, and it is clearly outperformed by the rest of the heuristic configurations. In particular, the average deviation of this configuration is 1.14%, highlighting the great instability of the heuristic approach. Moreover, from Figure 9, we see how this configuration reports the highest average deviation for all terminal-vessel combinations.

Next, we see how using a CP engine during the heuristic search can improve the performance of the solution method. The average deviations of both configurations are significantly reduced (0.6% and 0.81% for Configuration A and C, respectively), in addition to report the lowest minimum deviations (0.16% and 0.18%, respectively). The benefits of the collaboration of both search engines are clear. The heuristic explores the solution space, and when a promising area is found, the heuristic intensifies the search using CP techniques. Moreover, we also see how the particle swarm framework helps the math-heuristic to control the search for new solutions (Configuration A and B). Many local search heuristics are sensitive to the initial search point, and therefore, the particle swarm framework can add diversification to the search and prevent the heuristic being stuck in a non-promising area of the solution space. Furthermore, Configuration A, which combines the particle swarm framework with the CP engine within the heuristic search, reports the overall best performance. The results from Figure 9 show that Configuration A clearly outperforms

the rest of configurations by reporting a significant lower average deviation for almost all terminal-vessel combinations.

Finally, in Figure 10, we plot the performance of the ALNS math-heuristic during a run of the `PSP.4.12.3` instance. At the beginning, the heuristic initialises the search with a population of two different candidate points. During the first stages of the search, the heuristic accepts non-improving solutions with higher probability, allowing to explore large areas of the solution space. Once the heuristic finds a new best-known solution, the heuristic switches the search engine to use CP, and intensifies the search in the neighbourhood of the current solution. After the intensification phase, the particle swarm framework updates the population of the candidate solutions to continue the search from the new best found solution. This process is repeated until the math-heuristic reaches the maximum execution time. The acceptances criteria narrows the search during the last stages.

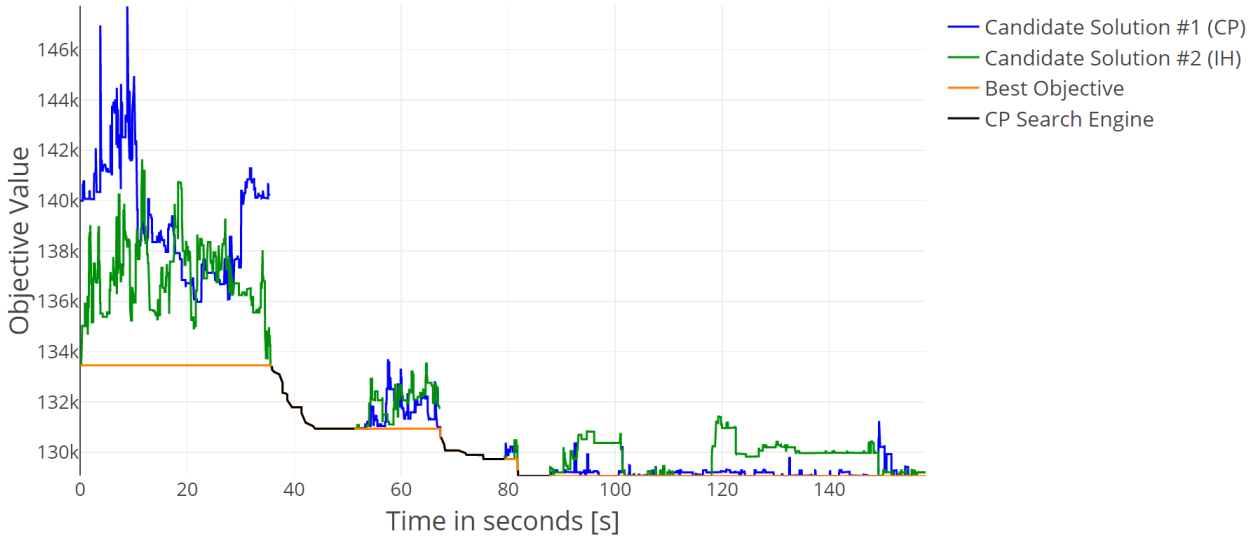

Figure 10: *Performance of the ALNS math-heuristic with particle swarm framework and CP engine within the heuristic search (Configuration A) for a run on the* `PSP.4.12.3` *instance.*

5 Conclusion

In this paper we presented different solution approaches for solving the PSP, the problem of scheduling the operations of feeder vessels in multi-terminal ports. Large ports, such as Rotterdam or Singapore, have excessive container traffic, and they represent the main transshipment points between container vessels. Generally, these ports have many terminals, where vessels can load and discharge containers. As liner vessels have large carrying capacities, they have priority when planning the port visits and they visit a single terminal in the port, where they discharge and load all containers designated to the port. On the other hand, feeder vessels handle cargo from multiple liner vessels, and they visit several terminals to transport all container to their corresponding destinations. The PSP studies the logistics and planning of

the feeder vessels. As a generalisation of the GSP, the PSP is $\mathcal{NP}$-hard and difficult to solve in practice for large instances. In this paper, we presented a compact CP formulation for the problem, and an efficient ALNS math-heuristic approach for solving large instances.

The CP formulation provides a more flexible modelling environment, which allows the user to easily add new side constraints to the problem without changing the overall structure. This is not always the case when developing metaheuristics, as a new constraint can sometimes make the basic structure of the heuristic invalid. We modelled and solved the problem using CPO. The extensive library of variables and constraints within CPO allows a compact and straightforward formulation of the problem. One of the biggest advantages is the possibility of modelling non-linear constraints without the burden of defining many *big-M constraints*, as needed in mathematical programming formulations. Time window constraints can be easily incorporated in the definition of the decision variables, and capacity constraints can be efficiently modelled using a variant of the one-to-many-to-one pick-up and delivery problem. CPO showed superior performance to other mathematical programming solvers such as CPLEX. In general, CPO found good quality solutions quite quickly, continuously improving very efficiently the current solution during the early stages of the search.

Furthermore, we should remark that the CP model only considered the handling of containers between the hub port and other ports. In some occasions, transshipment operations can also be performed between feeder vessels. These operations are infrequent, and they are similar to the pick-up and delivery operations in vehicle routing problems. In this case, the CP model can be easily extended to account for these operations. The model then requires the definition of dummy interval variables, whose domain and size are defined by the time windows of the pick-up and delivery operations. Finally, channelling constraints are needed to ensure the connections with the corresponding interval variables of the operations, and the cumulative function expressions must be extended with the *pulse* function for the new added variables.

However, the CP model cannot solve to optimality medium and large instances. For this purpose, we further proposed an efficient ALNS math-heuristic approach. We aimed to develop a solution method combining techniques from local search heuristics and CP in a meaningful way throughout the search. We proposed a math-heuristic framework where the search is performed using an ALNS heuristic, to efficiently explore the solution space. Then, when a new best-known solution is found, we intensify the search using CP techniques. Furthermore, we embedded the search with a particle swarm framework in order to add diversification during the heuristic search. We conducted computational experiments on the *PortLib* instances, showing that the combination of both methods can result in significant benefits. The proposed ALNS math-heuristic found overall best-known solutions as well as reported a more consistent average performance.

Finally, one of the reasons for developing and relying on heuristic methods for problem solving is that these methods can provide practical decision tools that can be used during the decision planning process. In general, heuristic approaches can return high-quality solutions relativity fast, without being necessary to perform optimality tests. Nevertheless, these methods are highly stochastic, and several runs are required to control the variance of the results. In this paper, we saw that the instability of the heuristic methods can be reduced by incorporating CP techniques during the search. The combination of both techniques

improved the performance of the heuristic, returning more stable and better quality solutions, in addition to providing information about the optimality of the solutions.

Compliance with Ethical Standards

Conflict of Interest The authors declare that they have no conflict of interest.

A Algorithm Procedure for the Initial Solution

We present the algorithm procedure for generating an initial solution. We propose a two-stage construction heuristic, which is further improved by a local search heuristic. The proposed algorithm is an extension of the algorithm procedure described in [15].

The procedure starts by generating an empty initial solution that only contains the dummy operations for the terminals and vessels from the set $O \setminus \tilde{O}$. During the first stage, the heuristic iteratively adds the interior operations into the empty initial solution using the greedy insertion method, i.e. the operations are inserted into the current solution in the best position that minimise the current objective value. The procedure continues to generate a family of initial solutions at the next stage. During the second stage, we identify and remove all infeasible operations from the current solution. Following this, we again use the greedy insertion method to insert all the previously removed operations. The last stage is then repeated until a feasible solution is obtained or until the current infeasible solution cannot be further improved.

We consider different orders in which the operations can be inserted during the second stage of the construction heuristic. The sorting rules, inspired by the dispatching rules for scheduling problems [26], are presented below. The operations are sorted by: (1) No sorting, i.e. operations are sorted as they are removed; (2) earliest start of time windows; (3) latest start of time windows; (4) increasing size of time windows; (5) increasing service times; and (6) decreasing service times.

The method creates an initial solution for each sorting rule, and then retrieves the solution with the lowest objective value as the initial solution. The pseudo-code for generating the initial solution can be seen below in Algorithm 2. Next, the initial solution is further improved by a simple local search heuristic, where the relocation of an operation is defined as a move in the neighbourhood.

Algorithm 2: Pseudo-Code for two-stage construction heuristic.

```
L ← Set of different sorting rules;
s ← Empty solution containing only non-interior operations from the set O \ Õ;
for Each interior operation p from Õ do
    Insert p in s using the greedy insertion method;
s* ← s;
for Each sorting rule l ∈ L do
    s' ← s;
    while stopping criteria is NOT met do
        Ô ← Infeasible operations from solution s';
        Sort operations from Ô using sorting rule l;
        for Each interior operation p from Ô do
            Insert p in s' using the greedy insertion method;
    if f(s') < f(s*) then
        s* ← s';
Return s*;
```

References

[1] Aldous, D. and Vazirani, U. (1994). "go with the winners" algorithms. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 492–501. IEEE.

[2] Baptiste, P., Le Pape, C., and Nuijten, W. (2012). *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media.

[3] Beck, J. C., Feng, T., and Watson, J.-P. (2011). Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23(1):1–14.

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

[5] Boschetti, M. A., Maniezzo, V., Roffilli, M., and Röhler, A. B. (2009). Matheuristics: Optimization, simulation and control. In *International Workshop on Hybrid Metaheuristics*, pages 171–177. Springer.

[6] Brucker, P. (1999). Scheduling algorithms. *Journal-Operational Research Society*, 50.

[7] De Backer, B., Furnon, V., Shaw, P., Kilby, P., and Prosser, P. (2000). Solving vehicle routing problems using constraint programming and metaheuristics. *Journal of Heuristics*, 6(4):501–523.

[8] Fleszar, K. and Hindi, K. S. (2018). Algorithms for the unrelated parallel machine scheduling problem with a resource constraint. *European Journal of Operational Research*, 271(3):839–848.

[9] Gedik, R., Kalathia, D., Egilmez, G., and Kirac, E. (2018). A constraint programming approach for solving unrelated parallel machine scheduling problem. *Computers & Industrial Engineering*, 121:139–149.

[10] Gerhards, P., Stuerck, C., and Fink, A. (2017). An adaptive large neighbourhood search as a matheuristic for the multi-mode resource-constrained project scheduling problem. *European Journal of Industrial Engineering*, 11(6):774–791.

[11] Gharehgozli, A. H., Roy, D., and de Koster, R. (2016). Sea container terminals: New technologies and OR models. *Maritime Economics & Logistics*, 18(2):103–140.

[12] Gökgür, B., Hnich, B., and Özpeynirci, S. (2018). Parallel machine scheduling with tool loading: a constraint programming approach. *International Journal of Production Research*, 56(16):5541–5557.

[13] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier.

[14] Grimes, D., Hebrard, E., and Malapert, A. (2009). Closing the open shop: Contradicting conventional wisdom. In *International Conference on Principles and Practice of Constraint Programming*, pages 400–408. Springer.

[15] Hellsten, E., Sacramento, D., and Pisinger, D. (2020). An adaptive large neighbourhood search heuristic for routing and scheduling feeder vessels in multi-terminal ports. *European Journal of Operational Research*. In press.

[16] Hojabri, H., Gendreau, M., Potvin, J.-Y., and Rousseau, L.-M. (2018). Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92:87–97.

[17] Kaveshgar, N., Huynh, N., and Rahimian, S. K. (2012). An efficient genetic algorithm for solving the quay crane scheduling problem. *Expert Systems with Applications*, 39(18):13108–13117.

[18] Kim, K. H. and Moon, K. C. (2003). Berth scheduling by simulated annealing. *Transportation Research Part B: Methodological*, 37(6):541–560.

[19] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[20] Kizilay, D., Eliiyi, D. T., and Van Hentenryck, P. (2018). Constraint and mathematical programming models for integrated port container terminal operations. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 344–360. Springer.

[21] Kovacs, A. A., Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579–600.

[22] Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250.

[23] Lin, S.-W. and Ting, C.-J. (2014). Solving the dynamic berth allocation problem by simulated annealing. *Engineering Optimization*, 46(3):308–327.

[24] Malapert, A., Cambazard, H., Guéret, C., Jussien, N., Langevin, A., and Rousseau, L.-M. (2012). An optimal constraint programming approach to the open-shop problem. *INFORMS Journal on Computing*, 24(2):228–244.

[25] Meisel, F. (2009). *Seaside operations planning in container terminals*. Springer.

[26] Montazeri, M. and Van Wassenhove, L. (1990). Analysis of scheduling rules for an fms. *The International Journal of Production Research*, 28(4):785–802.

[27] Pinedo, M. (2012). *Scheduling*, volume 29. Springer.

[28] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1):33–57.

[29] Qin, T., Du, Y., Chen, J. H., and Sha, M. (2020). Combining mixed integer programming and constraint programming to solve the integrated scheduling problem of container handling operations of a single vessel. *European Journal of Operational Research*.

[30] Qin, T., Du, Y., and Sha, M. (2016). Evaluating the solution performance of IP and CP for berth allocation with time-varying water depth. *Transportation Research Part E: Logistics and Transportation Review*, 87:167–185.

[31] Rifai, A. P., Nguyen, H.-T., and Dawal, S. Z. M. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*, 40:42–57.

[32] Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472.

[33] Sammarra, M., Cordeau, J.-F., Laporte, G., and Monaco, M. F. (2007). A tabu search heuristic for the quay crane scheduling problem. *Journal of Scheduling*, 10(4-5):327–336.

[34] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer.

[35] Stahlbock, R. and Voß, S. (2008). Operations research at container terminals: a literature update. *OR spectrum*, 30(1):1–52.

[36] Talbi, E.-G. (2016). Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, 240(1):171–215.

[37] Unctad (2018). Review of maritime transport. Technical report, United Nations Conference on Trade and Development.

[38] Unctad (2019). UnctadSTAT. `https://unctadstat.unctad.org/wds/ReportFolders/reportFolders.aspx`. [Online; accessed December 9, 2019].

[39] Watson, J.-P. and Beck, J. C. (2008). A hybrid constraint programming/local search approach to the job-shop scheduling problem. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 263–277. Springer.