An Adaptive Large Neighbourhood Search Heuristic for Routing and Scheduling Feeder Vessels in Multi-terminal Ports

Hellsten, Erik Orm; Sacramento Lechado, David; Pisinger, David

Published in:
European Journal of Operational Research

Link to article, DOI:
10.1016/j.ejor.2020.04.050

Publication date:
2020

Document Version
Peer reviewed version

An Adaptive Large Neighbourhood Search Heuristic for Routing and Scheduling Feeder Vessels in Multi-terminal Ports

Erik Orm Hellsten*[,a], David Sacramento[a], and David Pisinger[a]

[a] *Technical University of Denmark - Department of Technology, Management and Economics*

Abstract

This paper proposes an Adaptive Large Neighbourhood Search heuristic for solving the *Port Scheduling Problem*, the problem of scheduling feeder vessels' operations in multi-terminal ports. Each vessel has a number of operations to perform at different terminals within the port, and each terminal can only serve a single vessel at a time. The resulting problem is a general shop-like problem, with a variety of additional operational constraints. The objective is to let the vessels depart from the port as early as possible, as this allows them to sail at reduced speed to the next port, saving large amounts of fuel; as well as scheduling operations early, which leaves more slack for later and hence makes the system more robust. The developed Adaptive Large Neighbourhood Search heuristic works with the order of operations, and assigns the start times of the operations first as part of the solution evaluation. To conduct the computational experiments, a large set of benchmark test instances, denoted *PortLib*, was developed, and the performance of the heuristic was compared to that of a commercial solver. The results show that the heuristic in general finds better solutions, even with significantly shorter run times.

Keywords: OR in Maritime Industry, Port Operations, General Shop Scheduling, ALNS

1 Introduction

The maritime industry is a vital part of the global economy and there has in recent years been a large increase in the number of papers published on the topic of optimisation in maritime operations. Some of the main topics are liner shipping network design, stowage planning, and berth scheduling, for which advanced decision support tools are also increasingly being implemented in practice.

Today, the vast majority of international trade, around 90% by volume, is carried out by seaborne transportation. The containerised cargo trade roughly accounts for 23.8% of the total global trade, and the majority of the transport of this cargo is served by the liner shipping industry. Liner vessels are large vessels sailing on fixed itineraries, with weekly or bi-weekly frequency, and in 2019, more than 5,200 container liner vessels were in operation worldwide. The volume of containerised cargo has grown close to 8% per year during the last three decades, attaining an estimated volume of around 140 million TEUs (twenty-foot equivalent units). To satisfy this massive trade demand, the size of

*Corresponding Author.
Email addresses: erohe@dtu.dk (Erik Orm Hellsten), dsle@dtu.dk (David Sacramento), dapi@dtu.dk (David Pisinger).

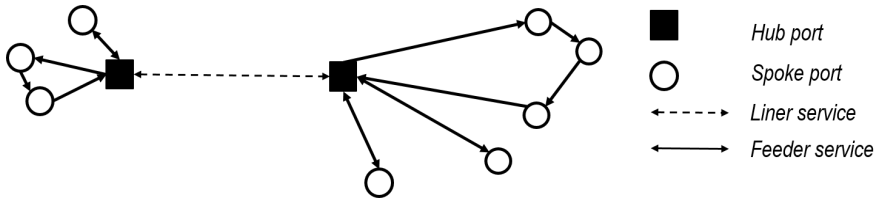

Figure 1: *Example of a hub and feeder structure in the shipping network.*

container vessels is increasing, and the largest vessels today can carry over 20,000 containers (Unctad, 2018, 2019). To facilitate the ever-growing liner vessel, each region typically has a few large ports, called *hubs*, where the liner vessels load and discharge containers. From the hubs, the containers are then transported to other smaller ports, also called *spoke ports* or *feeder ports*, by more flexible vessels, so called *feeder vessels*. Figure 1 depicts an example of a hub and feeder structure in the shipping network. Clearly, an efficient feeder structure is crucial for the liner shipping industry to operate effectively.

One of the drawbacks with this system, however, is that it heavily burdens the major hubs, as nearly all long-distance container traffic will be transshipped in those ports, between the liner vessels and the feeder vessels. To keep down the resulting congestion, as infrastructural improvements are huge capital investments and take significant time to implement, well-planned scheduling is of the essence to maximise the utilisation of the existing resources.

While feeder ports generally have only a single terminal, some large ports, such as Rotterdam or Singapore, are multi-terminal ports, i.e. they have several terminals that are located far apart. A terminal is a station within the port, where containers are loaded and discharged, using so called quay-cranes. As the liner vessels are very large, and have a significantly larger upkeep than the feeder vessels, they have priority in the planning process when visiting large ports. In practice, this means that every liner vessel visits only a single fixed terminal, where it discharges and loads all the containers for that port. Therefore, when planning the port stays of the feeder vessels, the terminal at which to load or discharge each container is already decided. Moreover, as each feeder vessel generally handles containers to and from multiple liner vessels, they need to visit multiple terminals within the port, especially as transporting containers between terminals by means of trucks or similar is expensive and should be avoided.

The port inspiring this work is Rotterdam, which is the largest hub in Europe and the gateway to the European market, with more than 500 visiting liner services connecting more than 1,000 ports all around the world. During 2017, more than 7,000 container vessels arrived to the port for loading and discharge operations, resulting in a total throughput of 142.6 million tonnes (Port of Rotterdam, 2017). Feeder and short-sea services play a very important role in the operations of the port of Rotterdam, and account for around 40% of the total container traffic. Figure 2 provides an illustration of the port of Rotterdam, where it can be seen how the container terminals are spread out along the river bank. In this case, the sailing time between the inner and outer terminals is around three hours.

Together with industry representatives, we have identified a new planning problem, namely that

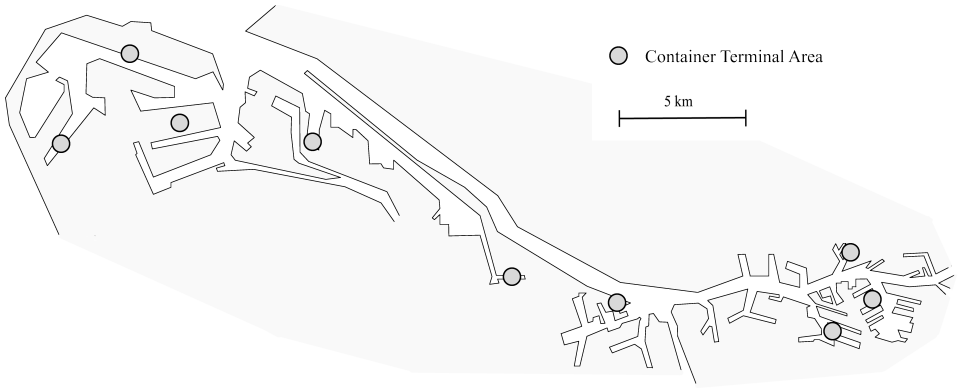

Figure 2: *Layout of the different container terminal areas in the port of Rotterdam.*

of routing and scheduling the feeder vessels through multi-terminal ports, deciding in which order to visit the terminals, and at which times. The main aim is to let vessels leave for the next port as early as possible, as sailing at a lower speed significantly lowers the fuel consumption, over the same distance. Additionally, we try to schedule operations early, to leave more resources available, in case of later changes, which makes the schedule more robust.

1.1 Problem Definition

Each feeder vessel visiting the multi-terminal port has a number of operations to perform at fixed, but usually different, terminals. The operations generally consist of either loading or discharging all the containers designated for a single terminal, and a vessel can have several operations assigned to the same terminal. In this way, a vessel can visit the same terminal multiple times during its stay at the port, if not all operations assigned to the terminal can be carried out consecutively. We also assume that the operations are non-preemptive, i.e. that they have to be completed without interruption once started, as this is almost always the case in practice. We use estimates for the arrival and latest departure for the vessels, for the travelling times between terminals and for the required time to perform the operations. The travel times and the arrival and departure times are normally given, whereas the operation times have to be estimated. We assume a known and given gross crane productivity for each operation, based on historical data, and estimate the operation time as the number of moves, i.e., the number of containers to lift on and off, to perform divided by this productivity. The gross crane productivity depends on the terminal, the vessel, the number of quay cranes active, etc. Generally, the terminal is, furthermore, contractually obliged to perform a minimum amount of moves per hour. More accurate operation times could likely be achieved by using an appropriate prediction algorithm, learning from historical data, but that is beyond this project.

The particular scope of this problem is based on an agreement between a single carrier, owning a fleet of vessels, and the terminal administration, which states that a number of terminals, with a set of accompanying quay cranes and workforce, are freely available to the carrier during certain time windows. The carrier is then free to schedule the operations of its feeder vessels at those terminals. Hence, in this case, the carrier is the owner of this problem. This way, the organisational interface

between the terminal and the carrier becomes significantly more clear-cut.

In the real-life case we have studied, the carrier was allowed to have a single vessel served at each terminal at a time, which is probably the most common scenario. In general, leasing the terminal for more time during the week allows for more flexibility than leasing space for multiple vessels simultaneously. Of course, if a carrier has a very large number of vessels, leasing space for multiple vessels would be necessary. However, we worked with one of the world's largest feeder companies, and so, such carriers are likely to be few. In some other cases, the physical layout of the terminal can also limit the maximum number of vessels to serve at the terminal.

Following this, we have assumed that each terminal can only serve one vessel simultaneously. It is of course possible, that in certain agreements, some of the terminals allow multiple vessels, and if there is no way to circumvent it, the carrier can sometimes manage to negotiate an additional berth. However, that is out of the scope of this work.

Most containers served in those large ports are being transshipped, i.e. they are being delivered to the port by one vessel and then picked up by another. Transshipments most commonly occur between a feeder vessel and liner vessel. For the feeder vessels this means that a load operation can first be performed once all containers are in place, and similarly a discharge operation has to be performed before any of the containers has to be delivered to another vessel. Again, as the connecting liner vessels are significantly larger, their schedule is created first and the schedule for the feeder vessels then has to adhere to that. Hence, this imposes time windows for the loading and discharge operations for the feeder vessels. We define the time window of an operation as the time period in which the operation must be started. Further, time windows are also defined for the vessels port stay. A vessel's time window begins at the arrival of the vessel to port, and ends at the latest point of departure at which the vessel will arrive on time to the next port on its itinerary, sailing at maximum speed.

A vessel may never arrive to the terminal after the end of the time window of the operation, but is allowed to wait until the start of the time window in a lay-by terminal close to the terminal of the operation. The availability of lay-by terminals is generally not binding and so, they are not considered in this work. The terminals can also have time windows; some terminals do not work during night hours or in the weekend, and others might not be open for the carrier in question during certain hours.

Additionally, we have a set of precedence requirements, defined by the carrier. The most common of those arise from the fact that a vessel has to discharge its containers at a terminal before it loads containers from the same terminal. Other precedence relations depend on for example the stowage plans, as to avoid unnecessary moves. In other more rare occasions, some containers need to be transshipped between two feeder vessels, in which the discharge of those containers has to be performed before they are loaded onto the next vessel.

We can define the *Port Scheduling Problem* (PSP) as finding an operational schedule for the feeder vessels, i.e. a starting time for each operation, which satisfies the above mentioned requirements as well as let the feeder vessels depart from the port as early as possible and packs the schedule as tight as possible to keep some slack for alterations and unexpected events. The fuel consumption

of a vessel is generally estimated to grow as the cube of the speed, which makes sailing slow highly tractable. Other costs, such as in-port fuel costs are marginal and are not considered in this work.

The PSP is a generalisation of the general shop problem, as defined in Brucker (1999), with time windows, capacity constraints and closing periods for the terminals. If the capacity at each terminal was unlimited, it would boil down to a set of single vehicle, 2-commodity, capacitated vehicle routing problems with time windows and precedence constraints. If instead, the order of terminals for each vessel was fixed, the problem would be a job shop problem with additional constraints. Further, the PSP is $\mathcal{NP}$-hard as it has the m-machine open shop problem as a special case.

The problem could be considered a sub-problem in the larger picture of optimising container-terminal operations. The schedules generated for the vessels will then be combined with quay crane schedules, stowage plans, yard operations, all the way down to worker schedules. Clearly, optimising the full operations is immensely complex, and so, dividing it into several steps seems to be an appropriate approach.

1.2 Literature Review

We will divide this literature review in two parts. In the first part we will briefly introduce the main literature on port operations and discuss how the PSP relates to other previously defined port operation problems. In the second part we will focus more on the mathematical structure of the problem and try to classify and compare it to other well-known optimisation problems.

For a detailed review on the topics of maritime optimisation, we refer the reader to the survey papers by Christiansen et al. (2013, 2019). There has been plenty of work on container-terminal operations, over which Stahlbock and Voß (2008), Meisel (2009) and Gharehgozli et al. (2016) have written excellent reviews.

In many regards, the *Berth Allocation Problem* (BAP) (Kim and Moon, 2003) is one of the most closely related problems. It serves essentially the same purpose as the PSP, which is to decide which vessel should occupy which berth at what time. The BAP appears when the terminal owns the problem of scheduling the berth allocation, where they have a number of available berths and a number of requested operations to perform on incoming vessels. In contrast, as mentioned before, the PSP appears when the carrier owns the problem, and as such the problems could be considered mutually exclusive.

The two other port-side problems generally considered in the container-terminal operation literature are stowage planning (Avriel and Penn, 1993), and crane-split and quay crane scheduling problem (QCSP) (Daganzo, 1989). There has been many studies on how to combine the BAP and the QCSP, for more efficient operations (Imai et al., 2008; Iris et al., 2017). But as the carrier is promised a fixed number of cranes, and there is only a single berth used at each terminal, performing the quay crane split and scheduling posterior to the port scheduling should be sufficient. The stowage planning, however, is deeply connected to the port scheduling, through the number of extra quay crane moves necessary due to overstowage. Solving an integrated stowage planning and port scheduling problem would, though, be prohibitively complex. So, in this work we are content with defining

a number of precedence relations between operations, preventing schedules which would require a large number of extra moves.

Another similar approach, application-wise, is the in-port routing of tanker ships by Wang et al. (2018). They developed a method to efficiently route a single tanker vessel through a large port with several operations at multiple terminals, far apart, with various tank allocation restrictions. The resulting problem, however, turns out to be widely different. The main property of the problem in Wang et al. (2018) is the tank allocation, whereas the defining property of the PSP is the limited availability of quay cranes which gives the PSP its scheduling characteristics. In contrast, in the tanker routing problem, only a single vessel is present, which makes it is a pure routing problem.

Structurally, the PSP is in essence a *machine scheduling problem* (Kan, 2012). More specifically, the PSP has a similar structure to the *General Shop Scheduling Problem* (GSP), as described in Brucker (1999). This is a more general definition of the shop scheduling problems which encompasses the well-studied *Open Shop Problem* (OSP) and the *Job Shop Problem* (JSP). Each vessel could be seen as a machine and each terminal could be seen as a job, such that each job has a number of operations that have to be performed at specific machines. Preemption is not allowed, and any machine can process at most one job at any time, and similarly, a job cannot be processed at more than one machine simultaneously. Additionally, there may be some precedence relations between the operations. The distances between terminals can further be formulated as sequence dependent setup times.

According to Graham's notation (Graham et al., 1979), the PSP can be roughly classified as $G_m|pred; r_p;\ d_p; s_{pq}|\sum w_p C_p$, i.e. an m-machine General Shop Problem with precedence constraints, release and due times for the operations and sequence dependent set-up times under an objective function which minimises the weighted sum of the completion time of the operations. However, the PSP is not a general shop scheduling problem, as this framework does neither include the closing periods for the terminals nor the capacity constraints for the vessels. Also, time windows for each individual operation is uncommon in the scheduling literature.

As the GSP has seen little attention in the literature, it is interesting to try to relate it to other more well studied scheduling problems. Among the classic machine scheduling problems, it has most similarities with the OSP (Blazewicz et al., 2019; Anand and Panneerselvam, 2015). While there are still significant differences between the OSP and the PSP, the base structure is similar and many of the solution concepts for OSP would also work for the PSP. A wide variety of heuristic approaches have been studied for solving both the classical OSP and different variants with additional constraints. The most common additional extra constraint, which is also present in the PSP, is the sequence-dependent setup times (Zhuang et al., 2019; Mejía and Yuraszeck, 2020; Abreu et al., 2020). Among the most successful approaches, we can highlight Genetic Algorithms (Abreu et al., 2020; Hosseinabadi et al., 2018) and Particle Swarm Optimisation (Sha and Hsu, 2008; Lin and Sha, 2011). Several authors have tried various hybrid genetic algorithms, combining a genetic algorithm with for example greedy heuristics (Kokosiński and Studzienny, 2007) and Tabu Search (Liaw, 2000).

Furthermore, in terms of local search neighbourhood-based heuristics, there are a number of

papers that use efficient methodology to solve different variants of the OSP, such as Simulated Annealing (Harmanani and Ghosn, 2016) and Variable Neighbourhood Search (VNS) (Mejía and Yuraszeck, 2020). The majority of the previous approaches use a permutation list of operations to efficiently model the problem. In the latter paper by (Mejía and Yuraszeck, 2020), for example, the authors use an efficient decoding scheme for the permutation list which is embedded in a self-tuning VNS for solving the OSP with travel times and sequence-dependent setup times.

Furthermore, the PSP can also be seen as a resource constrained project scheduling problem with sequence dependent setup times and time windows (Hartmann and Briskorn, 2010), but again with the addition of capacity constraints and closing periods. Nevertheless, this problem becomes more relevant for the PSP when the capacity of the terminals is no longer limited to serve a single vessel.

In this work we use an Adaptive Large Neighbourhood Search heuristic (ALNS) inspired by Pisinger and Ropke (2007). This heuristic has been successfully applied to several routing (Ropke and Pisinger, 2006; Pisinger and Ropke, 2007, 2010) and scheduling problems (Muller, 2009; Kovacs et al., 2012; Rifai et al., 2016).

1.3 Constraint Programming

One of the most successful methods to solve scheduling problems is *Constraint Programming* (CP), which has become the state-of-the-art method for many scheduling problems (Laborie et al., 2018). As an example, Malapert et al. (2012) reports the overall best performance for the classical OSP for a wide range of benchmark instances available in the literature. However, this is no longer the case for objective functions different from the make-span minimisation. IBM ILOG CP Optimizer provides a generic CP-based system to model and solve scheduling problem, which uses a constraint programming engine to prove optimality. We have chosen to benchmark against CPLEX, as the mathematical programming engine returns lower bounds and optimality gaps measures. Nevertheless, the implementation of CP tools for solving the PSP is an interesting subject for future work.

1.4 Contributions

The main contributions of this paper are:

- First, we present a new scheduling problem for feeder vessels in multi-terminal ports, which has been developed in close collaboration with industry. We model the problem as a *mixed-integer programming* (MIP) model, inspired by machine-scheduling formulations, and show that it is $\mathcal{NP}$-hard.
- Secondly, we propose an ALNS heuristic to find good solutions for the PSP, using a set of destroy and repair methods tailored for the problem. The ALNS framework uses weights for pairs of destroy and repair methods to promote pairs which work well together.
- Thirdly, we present a suite of benchmark instances, denoted *PortLib*, that have been generated with the aim of accurately representing realistic test cases. The benchmark instances serve as an instrument to measure and compare the performance of the proposed methods in the present

paper, as well as to provide a platform for future researchers to compare and develop further heuristics and exact methods.

- Lastly, we conduct computational experiments to find a good parameter setting and assess the performance of the heuristic. The results show that the ALNS heuristic provides good results within reasonable computation times, and that it outperforms CPLEX, which was used as benchmark.

The paper is organised as follows: Section 2 presents the mathematical model for the PSP. Section 3 is devoted to the ALNS heuristic and the main adaptions for this problem are described in detail. Section 4 provides a description of the *PortLib* instances. The results for the computational experiments are reported in Section 5. Finally, the paper is summarised and concluded in Section 6.

2 Mathematical Formulation

In this section we introduce the main notation for the PSP, which will be used to model the problem mathematically as a MIP. This provides an exact definition of the problem, and it is used for the benchmark. Appendix A gives an overview of all the notation used throughout the paper.

Let $\tilde{T} = \{1, \ldots, n\}$ be the set of n terminals in the port and define $T = \tilde{T} \cup \{0, n+1\}$, where 0 is the entry point of the port and $n+1$ is the exit point of the port. Let $\tilde{V} = \{1, \ldots, m\}$ be the set of feeder vessels to route through the port. Additionally, let us define $V = \tilde{V} \cup \{0, m+1\}$, where 0 and $m+1$ are dummy vessels for modelling purposes.

Further, let O_i^v be the set of operations to be performed by vessel $v \in V$ at terminal $i \in T$. We also define $O^v = \bigcup_{i \in T} O_i^v$, to be all the operations for vessel $v \in V$, $O_i = \bigcup_{v \in V} O_i^v$, to be all operations at terminal $i \in T$, and $O = \bigcup_{i \in T, v \in V} O_i^v$ to be the set of all operations. Here, O_i^0 and O_i^{m+1} are dummy operations denoting when terminal $i \in T$ starts working, respectively finishes, for the considered time period. Similarly, O_0^v and O_{n+1}^v represent the arrival and the departure, for vessel $v \in V$, to/from the port. Let further $\tilde{O} = \bigcup_{i \in \tilde{T}, v \in \tilde{V}} O_i^v$ be what we call the set of interior operations, i.e. operations which are not the dummy operations for the terminals or a vessel entering or leaving port.

Let δ_{ij} be the time it takes for a vessel to travel between the terminals $i \in T$ and $j \in T$. The travelling time between the same terminal, i.e., when $i = j$, is negligible and set to zero. Moreover, let w_p be the number of containers to be handled at operation $p \in O$. Positive values represent loading operations, whereas negative values represent discharge operations. Associated with the latter, let τ_p be the required time to perform the operation $p \in O$. We assume the gross crane productivities to be given and calculate the operation times as the number of required container moves divided by the gross crane productivity. Additionally, let λ_{pq} be binary precedence parameters, such that operation $p \in O$ has to be performed before operation $q \in O$ if $\lambda_{pq} = 1$. For each operation $p \in O$, denote the time window, in which the operation is allowed to start, by $[\alpha_p, \beta_p]$, and let ϕ_p and ν_p be the corresponding terminal and vessel of the operation. Note that $\alpha_{O_0^v}$ and $\beta_{O_{n+1}^v}$ denote the earliest arrival time and latest departure for vessel v to/from the port, respectively. Each vessel $v \in V$ has a maximum cargo capacity of Q_v containers, and arrives to the port with an initial cargo

of $\hat{q}_v$ containers. Lastly, let S_i denote a set of closing periods for terminal $i \in T$, within which no operations can be performed, and let $[\xi_s, \zeta_s]$ be the time window of closing period $s \in S_i$.

With all necessary sets and parameters defined, let us define variables y_p, representing the starting times of operations $p \in O$. As discussed in the introduction, the general objective is to depart early to save fuel, and schedule operations early to leave slack in the schedule for future changes. To model this we use the weighted sum of operation starting times as objective function,

$$f(y_p \mid p \in O) = \sum_{p \in \tilde{O} \cup O_{n+1}} c_p y_p, \tag{1}$$

where the sum over $\tilde{O}$ denotes the scheduled time of the interior operations and the sum over O_{n+1} denotes the departure times of the vessels.

What remains is to define the coefficients c_p. First, a problem with minimising the sum of start times in a schedule is that it strongly favours putting the short operations early. To avoid this, we weigh each interior operation $p \in \tilde{O}$ by its service time in the objective function. Secondly, various vessels have different priorities, and to model this we weigh each operation by a vessel specific priority factor, γ_v. Lastly, we use a coefficient, ρ, which denotes the relative importance between leaving early, and scheduling operations early. Hence, we get the following weights:

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

The values for ρ and γ_v have to be set according to the preferences of the carrier.

As in most machine scheduling problems we define binary decision variables representing the precedence between operations. For every distinct operation pair $(p, q) : p \in O, q \in (O_{\phi_p} \cup O_{\nu_p}) \setminus \{p\}$, let x_{pq} be a binary variable that takes value 1 if operation p precedes operation q. Let z_{ps} be a binary variable, which is 0 if operation $p \in \tilde{O}$ is performed before the closing period $s \in S_{\phi_p}$, and 1 if it is performed after. A MIP model of the PSP can then be formulated as follows:

$$\min \sum_{p \in \tilde{O} \cup O_{n+1}} c_p y_p \tag{2a}$$

$$s.t.\ y_q - y_p \geq \delta_{\phi_p \phi_q} + \tau_p - M_{pq}(1 - x_{pq}) \qquad p \in O, q \in (O_{\phi_p} \cup O_{\nu_p}) \setminus \{p\} \tag{2b}$$

$$x_{pq} + x_{qp} = 1 \qquad p \in O, q \in (O_{\phi_p} \cup O_{\nu_p}) \setminus \{p\} \tag{2c}$$

$$x_{pq} = 1 \qquad p, q \in \{O : \lambda_{pq} = 1\} \tag{2d}$$

$$\sum_{q \in O_v} w_q x_{qp} \leq Q_v - \hat{q}_v - w_p \qquad v \in V, p \in O_v \tag{2e}$$

$$\alpha_p \leq y_p \leq \beta_p \qquad p \in O \tag{2f}$$

$$y_p + \tau_p \leq \xi_s + \hat{M}_{ps} z_{ps} \qquad p \in \tilde{O}, s \in S_{\phi_p} \tag{2g}$$

$$y_p \geq \zeta_s - \bar{M}_{ps}(1 - z_{ps}) \qquad p \in \tilde{O}, s \in S_{\phi_p} \tag{2h}$$

$$x_{pq} \in \{0,1\} \qquad p \in O, q \in (O_{\phi_p} \cup O_{\nu_p}) \setminus \{p\} \tag{2i}$$

$$y_p \in \mathbb{R}^+ \qquad p \in O \tag{2j}$$

$$z_{ps} \in \{0,1\} \qquad p \in \tilde{O}, s \in S_{\phi_p} \tag{2k}$$

where

$$M_{pq} = \beta_p - \alpha_q + \delta_{\phi_p \phi_q} + \tau_p,$$

$$\hat{M}_{ps} = \beta_p + \tau_p - \xi_s$$

and

$$\bar{M}_{ps} = \zeta_s - \alpha_p.$$

The objective function (2a) minimises the weighted sum of starting times of operations. The sum over $\tilde{O}$ minimises the starting time of the interior operations, whereas the sum over O_{n+1} minimises the departure time of the vessels from the port. Constraints (2b) and (2c) define the disjunctive constraints for the operations. Constraints (2b) ensure that if operation q is scheduled after p, then q cannot be served before the completion of p, and Constraints (2c) ensure that operations for the same vessel or in the same terminal are not performed simultaneously. Moreover, Constraints (2d) define the specified precedence between operations.

The maximum cargo capacity must be respected at all times for each vessel during the scheduling of operations in the port, as expressed in Constraints (2e). For any operation of a given vessel, the current number of on-board containers is computed as the sum of the cargo handled by the vessel in the previous operations plus the initial load of that vessel. After serving an operation, the on-board containers shall not exceed the maximum cargo capacity of the vessel.

Constraints (2f) enforces the time windows of the operations. Additionally, an operation cannot be served during the closing periods of a terminal, as seen in Constraints (2g)–(2h). Constraints (2g) ensure that if an operation $p \in \tilde{O}$ is scheduled before a time period $s \in S_{\phi_p}$, the operation must be completed before the terminal becomes inoperative. Similarly, if the operation is scheduled after the time period, Constraints (2h) make sure that the operation does not start before the terminal becomes active again. Finally, the domains of the decision variables are defined in Constraints (2i)–(2k).

3 The ALNS Heuristic

Solving general shop-like problems with exact methods tends to be very time consuming. Hence, the more common approach to solve realistic-sized instances is to develop a metaheuristic. In this work, we propose a ALNS heuristic for solving the PSP. The ALNS framework has been successfully applied to several routing and scheduling problems (Pisinger and Ropke, 2010; Muller, 2009).

The ALNS heuristic was first introduced by Ropke and Pisinger (2006) as an extension to the Large Neighbourhood Search (LNS) heuristic previously presented by Shaw (1998). Starting from an initial solution, the heuristic progressively seeks for a new solution applying specialised heuristics to destroy and repair the current solution. Destroy methods are responsible for removing part of

the current solution, and normally, these methods contain some randomness to diversify the search for new solutions. Repair methods are defined with greedy strategies for the quick reconstruction of the current solution. The heuristic is encompassed in an adaptive framework, where the decision of which destroy and repair methods to use is based on the methods performance earlier in the run. In each iteration, a destroy method and a repair method are chosen based on their current scores. They are then applied to the current solution, and the heuristic decides whether to keep the new solution based on some acceptance criteria. Lastly, the scores for the methods are updated depending on the quality of the achieved solution. For a more general description of the ALNS framework, we refer to Pisinger and Ropke (2007). The main adaptions to this framework for this problem are described in detail in the following Subsections 3.1–3.9.

3.1 Splitting the Order Operations and Time Assignment

For a given order of operations, assigning optimal times for the operations can be done in polynomial time. This makes it possible to work with the order of operations rather than complete schedules and then assigning optimal times as part of evaluating a solution. This is commonly modelled with a disjunctive graph (Brucker, 1999), where a solution is represented by an acyclic orientation of the edges. The structure of our particular disjunctive graph is that the operations performed by the same vessel, as well as the operations served by the same terminal, are connected. The connections are made up of conjunctive arcs where there are precedence relationships and otherwise disjunctive edges.

As we are working with local search heuristics, we almost exclusively work with *orientations* of the disjunctive graph and it turns out that it becomes slightly cleaner if we instead work with the transitive reduction of the oriented disjunctive graphs. This corresponds to a graph where, for each vessel, the arcs form a path from the vessel's dummy start node to it's dummy end node, through each interior operation served by that vessel, and similarly for the terminals. We will denote this graph the *order graph*, as it represents in which order the operations are performed. The order graph has $|O|$ nodes, and $2|O| - 3(m + n)$ arcs, where m is the number of vessels and n the number of terminals. An example of an order graph can be seen in Figure 3. The squared nodes represent the non-interior operations from the set $O \setminus \tilde{O}$, whereas the circled nodes represent the interior operations $\tilde{O}$. Here. each vessel $v \in V$ visits each terminal $t \in T$ for discharging and for loading containers. The vertical lines show in which order the operations are performed at a given terminal, whereas the horizontal lines show in which order the operations are performed for a given vessel.

The order graph does theoretically not need to be connected, but if the graph is disconnected, each connected component represents a problem which could be solved independently. So let us for the remainder assume that the order graph is connected.

3.2 Assigning Time

Once we have an ordering of the operations, assigning the optimal operation times is a single-source shortest path problem with time windows in a directed acyclic graph, which is solvable in polynomial

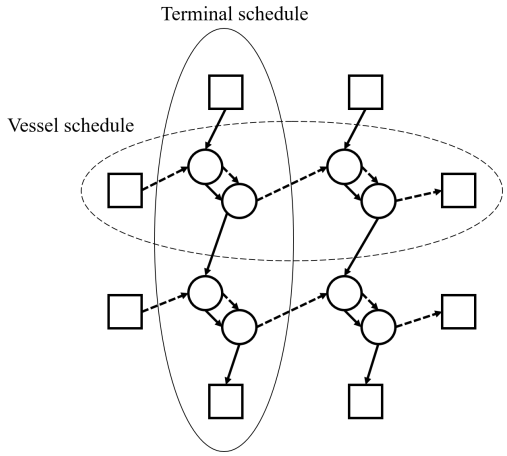

Figure 3: *Example of an order graph for an instance with two terminals and two vessels.*

time. We use a version of the classic algorithm, described in Section 24.2 in Cormen et al. (2009), modified to handle time windows.

The order graph provides a topological sorting of the operations. The earliest starting time for each operation can then be calculated through dynamic programming. First, the time of the first operations, i.e. the operations for the opening of the terminals and for the arrival of the vessels to the port, are assigned, and then, recursively, the times of the successors are assigned. Each time an operation's starting time is assigned, it has to be pushed forward as to satisfy the start of its time window and not collide with any closing periods. This procedure will create a semi-active schedule as the order graph is acyclic. This follows almost immediately from Theorem 24.5 in Cormen et al. (2009), with minor adaptions to include time windows and closing periods.

Once the times for all operations have been assigned, the algorithm checks if the time assignment is feasible with regards to the time windows and the capacity constraints, and add the corresponding penalties to the objective value.

3.3 The Destroy and Repair Methods

With the time assignment as part of the solution evaluation, the heuristic will work with orders of operations. In our heuristic, we make use of a few different destroy methods, and a number of repair methods, which follow various strategies when inserting operations. We define removing an operation from the order graph as removing it from the ordered set of operations of its terminal and of its vessel. Let $p \in \tilde{O}$ be the operation to remove, and let $\hat{O}$ be the set of removed operations. There exists one predecessor $p'_{\mathrm{T}} \in O_{\phi_p}$ and one successor $p''_{\mathrm{T}} \in O_{\phi_p}$ in the ordered set of its terminal, and predecessor $p'_{\mathrm{V}} \in O_{\nu_p}$ and one successor $p''_{\mathrm{V}} \in O_{\nu_p}$ in the ordered set of its vessel. Now, removing p from the order graph G, means that we remove the node representing the operation p along with the arcs connected to it, and then we add an arc from p'_{T} to p''_{T} and one arc from p'_{V} to p''_{V}, to keep the graph structure intact.

Inserting an operation $p \in \hat{O}$ means the exact opposite. We choose two operations $p'_{\mathrm{T}} \in O_{\phi_p}$ and

$p''_{\mathrm{T}} \in O_{\phi_p}$ from the ordered set of its terminal, such that there is an arc from p'_{T} to p''_{T}, and two operations $p'_{\mathrm{V}} \in O_{\nu_p}$ and $p''_{\mathrm{V}} \in O_{\nu_p}$ from the ordered set of its vessel, such that there is an arc from p'_{V} to p''_{V}. Now we remove those two connecting arcs, and instead add four arcs; from p'_{T} to p, from p to p''_{T}, from p'_{V} to p and from p to p''_{V}.

With removal and insertion defined, the destroy and repair methods are then defined as follows:

Destroy Methods:

D1. Random Removal

Remove b random operations from the order graph.

D2. Vessel and Terminal Removal

Remove all operations from a single vessel or a single terminal.

D3. Worst Removal

For each operation $p \in \tilde{O}$, compute the cost reduction achieved by removing operation p, normalised by its service time. Remove the b operations with the highest normalised removal cost reduction.

D4. Infeasible Removal

Remove b random operations as well as, with a probability of 50%, every infeasible operation.

Repair Methods:

R1. Greedy Insertion

For each operation $p \in \hat{O}$, in a random order, insert p where the resulting objective value is minimal.

R2. Sorted Greedy Insertion

Given an ordering of operations, for each operation $p \in \hat{O}$, insert p where the resulting objective value is minimal. The operations can be ordered according to three sorting rules: In decreasing duration, in decreasing size of its operational time windows or in decreasing end time of their time window.

R3. 2-regret Insertion

Iteratively insert the operation $p \in \hat{O}$, with the largest difference in objective value between its best and second best insertion, at its best position.

Note that many of the described destroy and repair methods cover a parametrised family of different methods. For example, D1 describes B different destroy methods, one for each value of $b \in \{1, \ldots, B\}$ and R2 describes three repair methods, one for each kind of sorting. In total, this constitutes $2B + 2$ destroy options, one for each value of B in D1 and in D3, and one for vessel removal and one for terminal removal in D2. D4 is used instead of D1 when the current solution is infeasible. Moreover, there are a total of five repair options, one for R1, three for R2, and one for R3.

The repair methods R1 and R2 all use the same greedy insertion, only varying the sorting of operations. R1 uses a random order, whereas R2 sort the operations according to the service time, the size of the time window or the time window end. The reason for using sorted greedy insertion is to schedule the most challenging operations first, as both operations with longer service time and tighter time windows tend to be harder to accommodate.

For the sake of completeness, we also test some neighbourhoods from the classic ALNS heuristic (Pisinger and Ropke, 2007, 2010) such as the destroy method D3 and the repair method R3. Regret

heuristics similar to R3 have further been used for solving the Vehicle Routing Problem with Time Windows (Potvin and Rousseau, 1993) and the Generalised Assignment Problem (Trick, 1992).

D3 is a destroy method, which iteratively selects the worst operation from $\tilde{O}$ to remove from the current solution. R3 is a 2-regret insertion, which computes the second-best and best position, in which to insert the operations, in the current partial solution, and inserts the operation at the position that maximises the difference between them. The method tries to prioritise the insertion of those operations that may incur a high cost if they are postponed.

While D3 and R3 give rise to more sophisticated neighbourhoods, they also are computationally more expensive. As the expensive part of the heuristic is the solution evaluation, the complexity is best measured in insertion trials. Let us assume that we should insert k nodes. The greedy insertion methods need $\mathcal{O}(|O|^2)$ insertion trials per operation to insert, and so runs in $\mathcal{O}(k|O|^2)$, while the 2-regret runs in $\mathcal{O}(k^2|O|^2)$. Further, D3 needs $\mathcal{O}(|O|)$ solution evaluations, whereas the other destroy methods do not need any.

3.4 Initial Solution

The heuristic requires an initial solution, and due to the strong dependency between operations, the time windows and the precedence constraints, finding good or even feasible initial solutions is non-trivial. In this section, we present the algorithm procedure for generating an initial solution.

The construction of the initial solution begins by defining an order graph G that only contains the non-interior operations, i.e., the set $O \setminus \tilde{O}$. First, the interior operations are iteratively added to the order graph using the greedy insertion method. Then any operations leading to infeasibility are identified, removed from the current solution and then re-inserted, again using the greedy repair method. The latter stage is repeated until no better solution can be obtained. 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.

3.5 Accepting New Solutions

The heuristic generates new solutions iteratively, and as the repair methods are heuristics, new solutions might be worse than previous ones. To avoid exploring areas of the solution space less likely to lead to good solutions, it is often a good idea to reject a new solution if it is worse than the previous one. The common practice in ALNS algorithms (Ropke and Pisinger, 2006), to balance exploration and exploitation, is to use the classic temperature based framework from Simulated Annealing (Kirkpatrick et al., 1983), where the probability of accepting a deteriorating solution decreases exponentially with time and with the decrease in objective value. We use this and define the temperature curve based on an initial and a final temperature which are given as the objective value of the initial solution times the factors T_{st} and T_f, respectively. As the acceptance probability is dependent on the quote between the change in objective value and the temperature, by scaling the temperature with the initial objective value, the heuristic becomes more stable with regards to variations in instance size. While the T_f factor is always small enough for the algorithm to reject

deteriorating solutions towards the end of the run, its exact value is still of great significance as it defines the shape of the exponential curve, for a given start value.

3.6 Adaptively Choosing a Destroy and Repair Method

As shown previously, there are a number of different destroy and repair methods to choose between, where each combination of methods could work well for different instances. Additionally, some methods may complement each other; larger neighbourhoods provide a lot of exploration and smaller neighbourhoods work better for exploitation. Since the nominal paper by Ropke and Pisinger (2006), it has been demonstrated many times that using an adaptive framework with probability weights for the different methods, is a well-working manner of balancing the usage of different destroy and repair methods.

The specific way to update probability weights differ between papers. It is common to promote methods which achieve local or global improving solutions, and demote methods resulting in deteriorating solutions. But it is also possible to promote exploration by for example rewarding solutions which have not been seen before.

In our framework, we use a weight for each combination of destroy and repair method, similar to what is done for example in Kovacs et al. (2012). This way, combinations of destroy and repair methods, which work well together, are promoted. The drawback, on the other hand, is that it generates significantly more weights to update, which leads to slower adaptation. An argument for using weights for pairs in this particular problem is that the algorithm runs sufficiently many iterations that the methods have ample time to adapt even the larger number of weights.

The update mechanism is then as follows: If a method pair finds an improving solution the weight is increased by a factor of π^+ and if it finds a worse solution it is decreased by a factor of $(1 - \pi^-)$, where π^+ and π^- are parameters guiding the adaptation speed. After each update, the weights are scaled such that they sum to one, since they represent probabilities.

3.7 Backtracking

To prevent getting stuck in local minima, and to intensify the search in promising areas of the solution space, we use a intensification mechanism common in many local search implementations (Gendreau and Potvin, 2019). If the heuristic fails to find a new global optimum after a fixed number of iterations, θ, then it returns to the best solution found so far. Furthermore, the temperature is increased to half the starting temperature, to encourage further exploration.

3.8 Infeasible Solutions

The heuristic allows infeasible solutions, but penalises them with a high cost. While the heuristic has yet to find a feasible solution, the general scheme of the temperature and acceptance of new solutions is modified. While the current solution is not feasible, the temperature feature is not activated. Hence, the acceptance criteria never rejects the new repaired solutions, allowing the heuristic to move freely to explore the solution space until a feasible solution is found. This proved significantly more efficient

for finding feasible solutions during initial testing. Further, the heuristic uses the destroy method D4, which also removes all operations that lead to infeasibility in the current solution, instead of the destroy method D1. Once the heuristic finds a feasible solution, the temperature feature is re-activated and the destroy method D4 is no longer used by the heuristic, and is instead replaced by D1.

3.9 Initial Constraint Propagation of Time Windows and Precedence Relations

As a pre-processing phase, we perform a series of constraint propagation techniques derived from Dorndorf et al. (2000). This phase narrows the time windows of operations and defines new precedence relations between the operations. The earliest and latest start time of operations are progressively tightened by applying several consistency tests. Note that while the new precedence constraints and tighter time windows generated by this process will never cut away any complete solutions from the solution space, it could tighten the LP-bound for the MIP and helps with discarding unfavourable operation placements during the ALNS repair methods. For example, a placement that seems promising while many operations are removed from the schedule, but which would lead to an infeasible solution once they are all inserted, could be disregarded. In practice it turned out to have a minor impact, however, partly because of that modern solvers would perform a similar procedure in its pre-processing phase.

4 Benchmark Instances

In this section we introduce the set of benchmark instances, which we denote the *PortLib* instances, developed to resemble real-life problems, which are used to conduct the computational experiments.

The problem we have proposed has been defined in close collaboration with representatives of the feeder line industry. While the PSP resembles other problems presented in the literature, the differences remain rather large. Hence, to test our methods we have generated a set of benchmark instances aiming at accurately representing realistic test cases. In addition to acting as a basis for us to do parameter tuning and test our methods, we hope that the instances can stimulate development of alternative heuristics and other variants of the problem.

The instances are named `PSP.n.m.r`, where `n` is the number of container-terminals, `m` is the number of vessels and `r` is the generic name of the scenario.

Based on the weekly operations of the studied feeder company in the port of Rotterdam, we consider instances with 2 to 5 terminals, and between 4 and 16 vessels. For each terminal-vessel combination, we assume a maximum of two operations, i.e. discharge and loading of containers, and we generate instances with 70% of the total maximum number of operations. This comes from the fact that normally not all vessels need to visit all terminals in the port, as it is almost always the case in real life. Moreover, in addition to the obvious precedence requirements for a vessel to discharge its containers before loading containers from the same terminal, a number of precedence constraints are added, corresponding to 10% of the total number of operations. These precedence may depend on the stowage plans and transshipment operations between feeder vessels. Other precedence relations can

be derived from more realistic examples, where the feeder company requires a vessel to first discharge containers at a specific terminal before visiting the remaining terminals.

Each operation is generated as a randomly selected combination of terminal and vessel, where the corresponding type (discharge or loading of containers) is randomly assigned. The required service time τ, in hours, follows a uniform distribution $U(2, 10)$, and the number of containers w to be handled at the operation is then calculated assuming a fix gross crane productivity of 20 containers per hour, which is based on historical data on the average productivity of the terminals. In each instance roughly 15% of the operations are subject to strict time windows.

We will assume a fleet of small feeder vessels with varying maximum cargo capacity from 500 to 1,000 TEU (Brouer et al., 2013). In addition, we assume that vessels arrive at the port relatively loaded with containers, typically above 70% of its cargo capacity, something which is almost always the case in reality. A high or low priority will be randomly assigned to each vessel of the fleet with equal probability. The time windows for the vessels are generated so they arrive within the first days of the planning horizon and they have sufficient buffer time to service their operations in the assigned terminals.

In terminals such as Rotterdam, the sailing time between the farthest terminals is significant, and up to 3 hours. Therefore, we randomly place the terminals in a grid of dimensions $\lceil \sqrt{n} \rceil \times \lceil \sqrt{n} \rceil$, where the pilot station, representing the entry/exit gate of the port, is always placed in one of the corners. We use the Manhattan norm as the distance function between terminals. This also ensures that the triangle equality holds. Moreover, each terminal can have up to two closing periods with a maximum duration of 15% of the total service time of the busiest terminal.

Following the description above, we have generated two sets of benchmark instances: The *training instances*, which were used to conduct the parameter analysis, and the test instances, denoted *PortLib*, which were used to evaluate the performance of the tuned heuristic. The sets are distinct, but have the same characteristics. The *PortLib* instances will be published online at `Zenodo.com` [1]. Additionally, a smaller set of instances were generated for the sensitivity analysis.

The generated instances are constructed to be realistic, but in addition we ensured that each instance has a feasible solution as well as strove towards that each constraint should have a significant impact. In general, the instances are made to be slightly harder to solve than the problems faced by industry, in order to challenge the developed methods, as well as spurring further development. Both the benchmark sets each contains 300 randomly generated instances grouped into 15 terminal-vessel combinations. A brief summary of the instances is shown in Table 1.

Table 1: *Overview of the PortLib instances. The instances are named* `PSP.n.m.r`*, where* `n` *is the number of container-terminals,* `m` *is the number of vessels and* `r` *is the generic name of the scenario.*

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

[1] https://doi.org/10.5281/zenodo.3760979

5 Computational Experiments

To study the computational performance of the ALNS heuristic, we compare it to solving the MIP model (2a)–(2k) using a commercial MIP-solver. As a solver we used CPLEX version 12.9 limited with a maximum execution time as specified in the following Section 5.1. Moreover, the ALNS heuristic has been implemented in Java, and both methods were run on a Huawei XH620 V3 computer with a 2.6GHz Intel Xeon Processor 2660v3. The parameter analysis was conducted on the *training instances*, whereas the test performance and comparison of the proposed methods were conducted on the *PortLib* instances.

One of the main factors of almost every metaheuristic, is a good parameter setting. We therefore first present experimental results for the analysis of the main parameters of the heuristic. Next, the ALNS heuristic and the MIP model solutions are compared. Lastly, a sensitivity analysis for the main constraints is conducted.

5.1 Run Times

The inherent trade-off between run time and solution quality is a crucial question for almost every optimisation algorithm. What to aim for depends on the use case for the heuristic, which for the PSP is twofold. The first use case is the operational planning of the daily schedule, which also includes remaking the schedule in response to potential disruptions. In this case the heuristic has to run in a few minutes as to not be a hindrance in the planner's work. The second use case is the strategic planning, where the aim is to make a recurring plan, which is relevant as liner vessels sail on weekly itineraries. Here, the quality of the solution is more important than the run time, and so longer run times are accepted.

In an attempt to balance run time and performance, we decided to test the heuristic for the two use cases. As the instances vary significantly in size, for the short run times we decided to let the run times increase with the size of the instances. We decided to use run times roughly proportional to the third power - more precisely, for an instance with $|\tilde{O}|$ operations, the run time would be $\max(5, \lceil |\tilde{O}|^3/2000 \rceil)$ seconds. The precise run times can also be seen in Table 2. For a fair comparison in the computational experiments, we decided to use a maximum time limit for CPLEX of 10 times that of the heuristic, for each instance. For the long run times, we only study instances with 5 terminals and there we instead used one hour for the heuristic, and 10 hours for CPLEX.

Table 2: *The short run times for the heuristic and for CPLEX for the different sizes of instances.*

Terminals	2			3				4				5			
Vessels	4	6	8	6	8	10	12	8	10	12	14	10	12	14	16
ALNS [s]	5	5	7	9	20	38	67	46	88	158	247	172	297	471	703
CPLEX [s]	50	50	70	90	200	380	670	460	880	1580	2470	1720	2970	4710	7030

5.2 Parameter Analysis

In this section we aim to study the effect of the different parameters, as well as the different destroy and repair methods, on the performance of the heuristic. Both to increase the understanding of how the heuristic works and to find a well-performing parameter setting.

Due to the large computational time required to carry out all the experiments, we decided to perform the parameter analysis mainly using short run times on the *training instances*. However, as the run time is likely to significantly affect the performance of additional methods, we further tested the effect of including extra methods for long run times on the best parameter setting found for short run times, which is discussed in Section 5.3.1.

5.2.1 Parameters

The main parameters to study are those controlling which destroy and repair methods to use. The *random removal* (D1) and the *greedy insertion* (R1) are considered the core methods, and they will always be used by the heuristic.

The parameter B denotes the maximum operations to remove in the random removal destroy method. A high number of removals opens up for a larger search space, but can also challenge the repair method, and lead to increased time consumption in each iteration.

The parameters T_{st} and T_f, are the start and end temperature factors, which will then be multiplied by the initial solution value to get the initial and final temperature. Further, θ, denotes the number of iterations without finding any improving solutions before the algorithm resets to the best found value.

For the update factors for the repair and destroy weights, π^+ and π^-, we have decided to fix the penalty values to $\pi^+ = 1\%$ and to $\pi^- = 0.5\%$, as this was providing good results in the phase of initial testing.

5.2.2 Performance analysis of the ALNS heuristic

After a phase of initial testing to get a good initial parameter setting, we decided to study and adjust the parameters individually. While the effect of the parameters cannot be expected to be independent, to jointly study or tune the parameters become exhaustively time-consuming, due to the inherent stochasticity of the problem. A decoupled study of the parameters also adds valuable insight into each individual parameter's impact on the heuristic.

To be able to compare results over different instances with various objective values, we have in all our experiments used percent over the best solution found, as the metric. This way we can average results over different instances in a meaningful way.

The most important parameter is the maximum number of removals in the random removal destroy method. As it would make little sense to use the same number of removals for an instance with 10 operations as for one with 200, we are interested to see how different settings perform for instances with different numbers of operations. We then study the effect of this parameter on the heuristic's performance, aiming to define the maximum removals as function of the instance size.

Table 3: *Average percentage deviation from the best solution found for the different values of the temperature factors. The rows correspond to different values for the start temperature factor, and the columns to different values for the end temperature factor.*

		End Temperature Factor		
		0.01	**0.001**	**0.0001**
Start Temperature Factor	**0.1**	1.658	1.234	1.342
	0.05	1.547	1.109	1.267
	0.025	1.405	1.079	1.271
	0.01	1.201	1.148	1.435

Each instance was run 10 times for each value of $B \in \{2n+1 \mid n = 0, \ldots, \sqrt{O}\}$ and then a second order polynomial was fitted to the best setting found for each instance size tested. The resulting polynomial, over an interpolation of the average gap for different maximum number of moves for different numbers of operations is shown in Figure 4. We see that the heuristic performs poorly when B is too low, but is less sensitive when B becomes larger. This is expected, as the heuristic is embedded in an adaptive framework, and can as such, to some extent, choose the number of removals which best fit the current instance. We further see that the average gap increases significantly for larger instances.

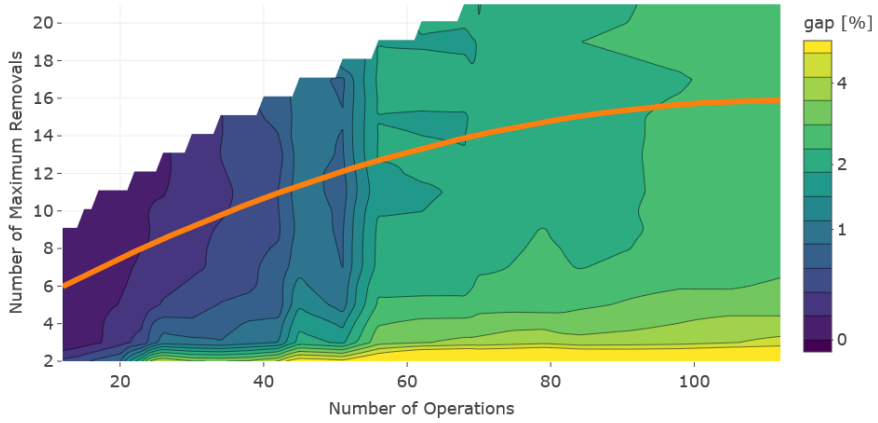

Figure 4: *A contour map over how well the method performs for various numbers of maximal removals and various number of operations with the fitted second order polynomial.*

Next, the start and end temperature factors, T_{st} and T_f were studied. Following a similar approach, each instance was run five times for each combination of parameters, and the results are shown in Table 3. We see that a start temperature of 2.5% of the initial objective value and an end temperature of 0.1% of the initial objective value yielded the best results.

Further, we see that when we have a high start temperature, it is better to have a low end temperature, and when we have a low start temperature, it is better to have a high end temperature. A possible explanation is that it is important to have elements from both exploration and exploitation, and that when both the start and end temperature factors are either too low or too high, the heuristic becomes too focused on a single one of them.

In Table 4 we see the average gap from the best solution found when introducing the reset

Table 4: *Average percentage deviation from the best solution found for the different values of the reset parameter.*

Reset Parameter	Av. Gap [%]
800	1.139
1,600	1.047
3,200	1.031
6,400	1.020
∞	1.095

Table 5: *Summary of the results obtained by the ALNS heuristic with different combinations of repair and destroy methods. For each of the tested destroy and repair methods the average gap to the best solution found is presented. In each combination, in addition to the presented methods, the random removal (D1) and greedy insertion (R1) methods are also used.*

Additional Methods		Gap [%]
None		1.016
Sorted Greedy Insertion	R2	0.999
2-regret	R3	1.414
Terminal and Vessel Removal	D2	1.112
Worst Removal	D3	1.024

mechanism, for a few different values of θ. We see that the reset method has a clear improving effect on the heuristic and that the performance increases progressively with the value of the reset parameter. Resetting the current solution to the best known solution after 6,400 iterations without improvement seems to yield the best performance, out of the tested values. The reason for that no higher values were tested is that θ is reaching towards the total number of iterations performed for the largest instances, which is around to 10,000.

Lastly, we studied the effect of including the remaining repair and destroy methods, presented in Section 3.3. In Table 5, we see the average gap when running the heuristic 10 times on each instance, with the different additional methods included. We see that the heuristic performs well when considering exclusively the base case, and an improvement with the addition of the repair method R2, which is the greedy insertion with different sorting strategies. Yet, the differences are rather small, which is expected as each of the extra methods only is one of many repair and destroy methods in an ALNS framework. Moreover, in trying several combinations of the repair method R2 with the remaining methods seemed to not yield any further improvements.

It is clear from the results that, for this parameter setting, the performance of the heuristic is better when considering the fast repair methods, which allow a greater number of iterations, when using the short run times. Nonetheless, in Section 5.3.1, we will see that the more elaborate methods perform better for longer run times.

After iteratively improving the parameter values from an initial good parameter setting, the final parameter setting which will be used in the computational experiments is summarised in Table 6.

Table 6: *Final parameter setting for the ALNS heuristic with short run times.*

Parameter	Value
B	$\lceil -0.0013\lvert\tilde{O}\rvert^2 + 0.25\lvert\tilde{O}\rvert + 2.89\rceil$
T_{st}	0.025
T_f	0.001
θ	6,400
Destroy Methods	D1
Repair Methods	R1, R2

5.3 Computational Results

The complete results for all the test instances for both CPLEX and the ALNS heuristic can be found as a supplement to this paper and online at `Zenodo.com` [2]. The results from solving the *PortLib* instances, using CPLEX, are summarised in Table 7. The instances are grouped by number of terminals, and then further divided by the number of vessels visiting the port. The smaller instances, with less than 26 operations, were solved to optimality in a few seconds. We also see that instances with around 40 operations seem to be the upper limit for what CPLEX can solve to optimality within the given time frame. Note further, how different instances of the same size vary significantly in difficulty. For the instances that reach the maximum execution time, the resulting optimality gaps are rather high, as shown in the lower part of the table, reaching up to 50% for the largest instances. For the remainder of this section, we will report as the CPLEX solution the objective value of the best integer solution found after reaching the maximum CPLEX execution time for the given instance.

Table 7: *Summary of the CPLEX results for all PortLib instances. Instances are grouped by number of terminals and number of vessels. The table reports the total number of operations, the number of optimal solutions, the average execution times in seconds (s), the average optimality gap in percentage (%), and the interval optimality gap of the instances. The results for the instances reaching the time limit are denoted with (t.l).*

Instance Name	Operations	Optimal	Avg. CPLEX Time (s)	Avg. Optimality Gap (%)	Interval Gap (%)
`PSP.2.4.r`	12	20/20	0.0	0.0	[0.0,0.0]
`PSP.2.6.r`	17	20/20	0.1	0.0	[0.0,0.0]
`PSP.2.8.r`	23	20/20	1.4	0.0	[0.0,0.0]
`PSP.3.6.r`	26	20/20	2.6	0.0	[0.0,0.0]
`PSP.3.8.r`	34	17/20	64.7	1.8	[0.0,14.8]
`PSP.3.10.r`	42	7/20	286.0	9.5	[0.0,30.8]
`PSP.3.12.r`	51	1/20	651.5	20.5	[0.0,35.3]
`PSP.4.8.r`	45	4/20	404.8	10.2	[0.0,22.5]
`PSP.4.10.r`	56	0/20	t.l	23.7	[8.9,39.5]
`PSP.4.12.r`	68	0/20	t.l	31.2	[17.8,42.7]
`PSP.4.14.r`	79	0/20	t.l	35.3	[24.8,45.0]
`PSP.5.10.r`	70	0/20	t.l	27.7	[18.0,36.7]
`PSP.5.12.r`	84	0/20	t.l	34.1	[21.9,43.0]
`PSP.5.14.r`	98	0/20	t.l	38.1	[29.3,52.1]
`PSP.5.16.r`	112	0/20	t.l	38.8	[28.1,47.1]

We run the ALNS heuristic 10 times on each of the *PortLib* instances with the best configuration found during the parameter analysis process and using the time limits from Table 2.

[2] https://doi.org/10.5281/zenodo.3760979

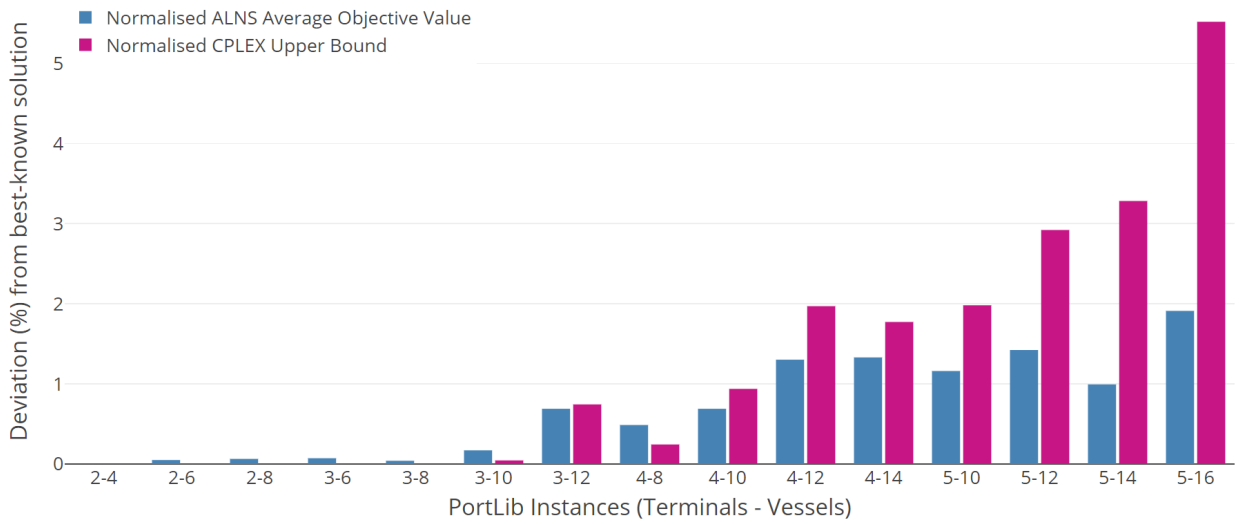

Figure 5: *The bar plot depicts the average objective value for the heuristic and the objective value by CPLEX, aggregated by the number of terminals and number of vessels, after being normalised with respect to the best-known solution obtained from the both methods.*

In Figure 5, we see the results for the *PortLib* instances, which have been normalised and aggregated by the number of vessels and terminals. In particular, the heuristic proves effective for larger instances with more than 50 operations, where its average performance is consistently better than the solution obtained from CPLEX. As shown in Figure 5, the average performance of the heuristic does not deviate excessively from the best-known solution as the number of operations increases; the difference between the average and best objective values always stay below 2%. The difference between the CPLEX solution and by the best-known solution increases more drastically, reaching more than 5% for the largest instances.

Next, we look at the difference for individual instances. In Figure 6, for each instance, we plot the quotient of the average ALNS objective value and the CPLEX objective value. The graph is constructed so that, in each section, the instances are ordered after the number of vessels, where instances with more vessels are further to the right. We see that the ALNS heuristic in general outperforms CPLEX, using significantly less time. As would be expected, we see that the heuristic is performing better, relative to CPLEX, on larger instances. For the smaller instances, where we know that CPLEX has found the optimal solution, we see that the heuristic quite consistently finds the optimal solution in each of the 10 runs, with only a few outliers. In the graph, those would be the instances up until half-way through the section with 3 terminals.

If we instead look to Figure 7, we see the quotients between the best solution of the ALNS over the 10 runs and the CPLEX solution. In this comparison, the heuristic is consistently better, and for the largest instances the differences reaches 10%. We also see that the differences between the the best solutions and the average solutions are not that large, which we also saw in Figure 5. Finally, to further clarify, the results are summarised in Table 8.

To further study the performance, we show the results for the `PSP.5.16.r` instances in more

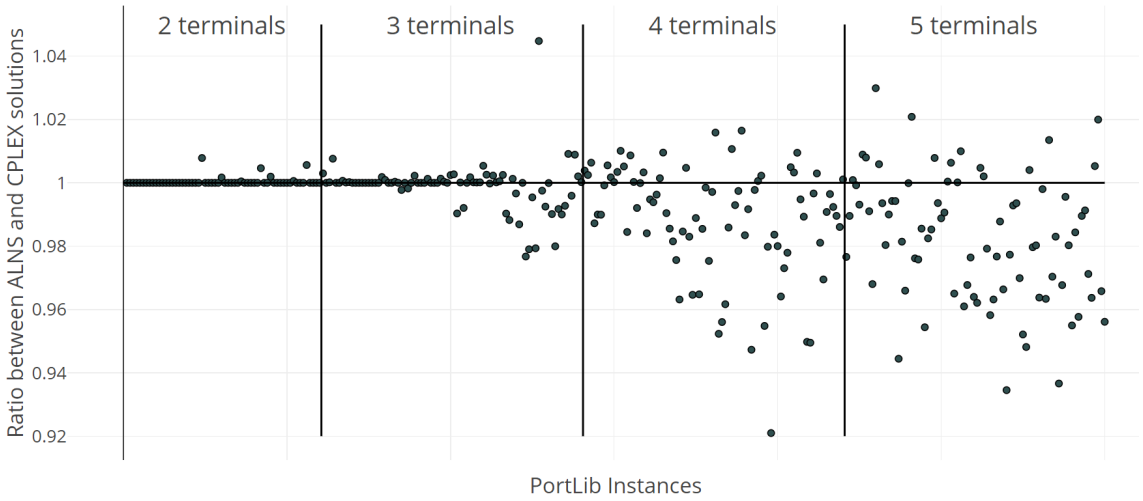

Figure 6: *Ratio between the average performance of the ALNS heuristic and the CPLEX solution for each of the* PortLib *instances, using the time limits from Table 2. A value below 1 means that ALNS was able to find a better solution than the solution from CPLEX.*

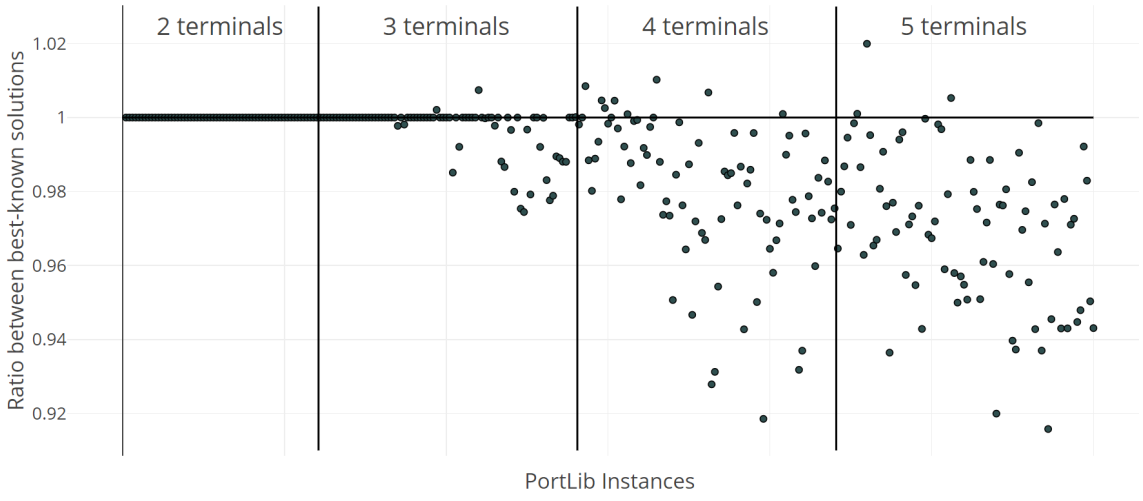

Figure 7: *Ratio between the best solutions found by the ALNS heuristic the CLPEX solutions for each of the* PortLib *instances, using the time limits from Table 2. A value below 1 means that ALNS was able to find a better solution than the solution from CPLEX.*

Table 8: *Summary of the results divided by number of terminals for the ratios between the best and average solutions, respectively, found by the ALNS heuristic and the solution from CPLEX.*

Ratio r	**2 Terminals**		**3 Terminals**		**4 Terminals**		**5 Terminals**	
	Average	**Best**	**Average**	**Best**	**Average**	**Best**	**Average**	**Best**
$r < 1$	0	0	22	23	55	69	63	77
$r > 1$	7	0	32	3	25	8	17	3
$r = 1$	53	60	26	54	0	3	0	0

Table 9: *Results for the* `PSP.5.16.r` *instances. Reported are: the best and average value found by the ALNS heuristic together with percentage error of the discrepancy between the average and best value found by the ALNS heuristic, and the best integer value found by CPLEX.*

Instance Number	ALNS Best	ALNS Average	Percentage Error	CPLEX Best
1	306,831	311,669.1	1.58	312,228
2	217,592	222,339.5	2.18	230,796
3	213,511	216,723.8	1.50	213,835
4	236,437	244,857.3	3.56	252,336
5	266,035	269,238.0	1.20	273,893
6	241,263	246,755.1	2.28	263,450
7	301,739	308,827.4	2.35	319,136
8	221,382	225,710.0	1.95	226,712
9	249,949	254,266.8	1.73	259,384
10	350,639	355,099.5	1.27	371,843
11	281,495	283,333.5	0.65	287,836
12	231,876	235,485.4	1.56	245,892
13	281,589	286,959.5	1.91	289,990
14	235,009	239,518.8	1.92	241,622
15	301,210	309,660.0	2.81	318,835
16	335,194	340,773.0	1.66	353,614
17	303,141	307,152.7	1.32	305,538
18	238,994	248,000.6	3.77	243,146
19	298,500	303,361.2	1.63	314,108
20	258,593	262,173.0	1.38	274,204

detail in Table 9. Here, the upper bound from CPLEX is presented, alongside the best and average objective values for the solutions found by ALNS. We see that the best solutions found by ALNS are significantly better than the best solutions found by CPLEX, and also that the average is consistently better. There is, on the other hand, a non-negligible discrepancy between the best and average solutions found by the ALNS heuristic. To some extent, this might be due to the fact that the problem consists of large discrete blocks and a single change may have a large impact on the solution value.

5.3.1 Experiments with long run times

To study the performance of the heuristic as a tool for strategic planning, we performed a number of experiments using a run time of one hour for the ALNS heuristic and 10 hours for CPLEX. First, we are interested in the performance of the destroy and repair methods in this new setting. For this experiment, we considered all instances with 5 terminals from the *PortLib* instances. We ran

Table 10: *Summary of the results obtained by the ALNS heuristic with different combinations of repair and destroy methods and a run time of 3,600 seconds.* ***Conf.:*** *Configuration number of the methods.* ***Av. Gap:*** *The average gap to the best found solution for the tested combinations of destroy and repair methods. For each configuration the random removal (D1) and greedy insertion (R1) methods are also included as a base case. The table reports first the results for the addition of the methods individually, and then, the remainder of the table is sorted by decreasing average gap.*

	Repair Methods		Destroy Methods		
Conf.	R2	R3	D2	D3	Av. Gap [%]
Base					1.719
1	×				1.493
2		×			1.417
3			×		1.665
4				×	1.609
5		×	×	×	1.744
6			×	×	1.704
7		×		×	1.643
8	×	×		×	1.612
9	×			×	1.500
10	×		×		1.499
11	×	×	×	×	1.488
12	×		×	×	1.463
13	×	×	×		1.447
14		×	×		1.388
15	×	×			1.380

the heuristic 5 times on each instance for all of combinations of the destroy and repair methods. We see from the results in Table 10 that, with the increased run time, all specialised methods now individually improve the base configuration of the heuristic.

How the methods work together is intricate, but in general we see that, combinations including the worst removal (D3) seem to perform worse and combinations including the sorted greedy repair method (R2) performs well. The 2-regret (R3) seems to perform well, but not together with the worst removal. In the light of the good performance of the sorted greedy insertion, it seems a little surprising that configuration 14 performs so well. The move terminal and move vessel neighbourhood (D2) constitutes only two destroy methods, and the impact of adding this neighbourhood is less. This can be seen by that the configurations which uses D2 together with another method, performs similarly to the same configurations, but where D2 is excluded. In the end, the configuration which includes the two other repair methods to the base configuration gave the best result for the long run times.

While the results necessarily are affected by the stochastic nature of the heuristic, the results presented in Table 10 are consolidated from 6,400 hours of run time. Taking this into consideration, it seems unlikely that the results would change significantly by running more experiments. We can conclude that, when giving more time to the heuristic, the search for new solutions can be carried out more effectively by considering more specialised neighbourhoods, and that the most efficient setting for strategic planning is using both R2 and R3, apart from the core methods D1 and R1.

In Figure 8, we show the results for each of the `PSP.5.12.r` *PortLib* instances. The graph shows

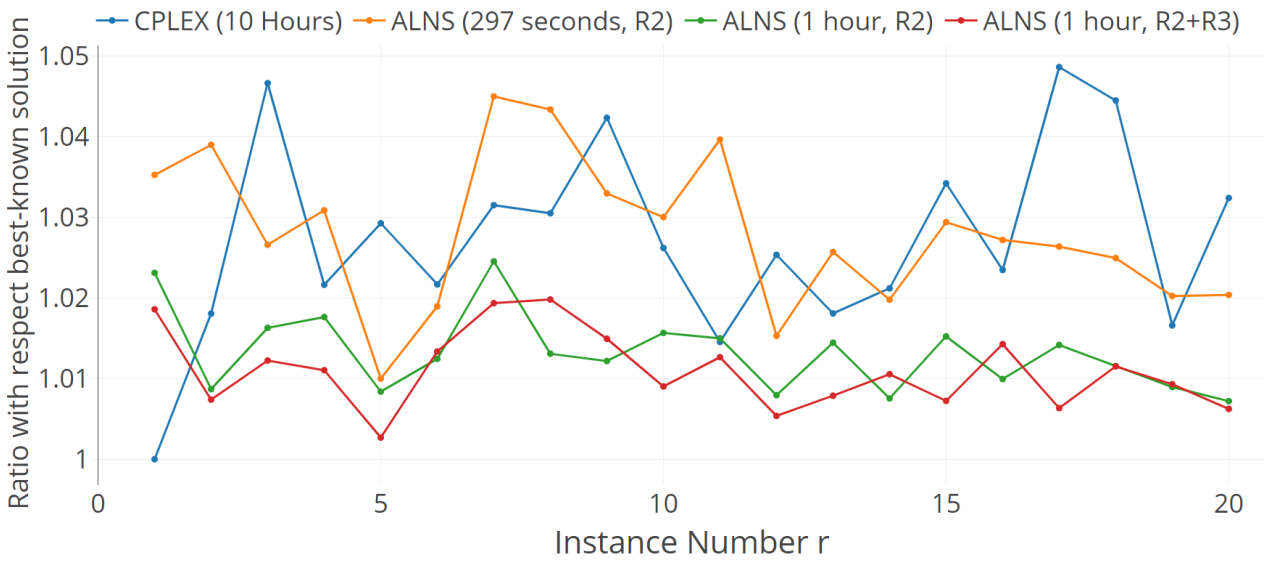

Figure 8: *The graph shows the average performance of the ALNS heuristic for a number of configurations, divided by the best objective value found, for the 20 PortLib instances in* `PSP.5.12.r`.

the results for the ALNS using the long run times of one hour with the best configuration for the short run times as found in Section 5.2, denoted 'ALNS(1 hour, R2)', as well as with the best configuration for the long run times as discussed above, denoted 'ALNS(1 hour, R2+R3)'. For reference, it also contains the results for the ALNS using the standard short run time of 297 seconds with the best configuration for short run times, which is denoted 'ALNS(297 seconds, R2)'. Lastly, the graph shows the objective value found by CPLEX after 10 hours of execution time. In each case, the objective is normalised by the best value found, and for the ALNS, each configuration is run 10 times on each instance, and the average ratio from the best found solution is shown.

There are a few key observations to make. First, we see a significant improvement in the results when running the heuristic for a longer time. The average results are not only much better, but also significantly more consistent. We see that even with the short run times, the heuristic still outperforms CPLEX on some instances, even though it here uses less than 1% of the run time of the solver. For the long run times, when the heuristic uses 10% of the run time of CPLEX, the heuristic performs clearly better.

Comparing the change in objective value for CPLEX when increasing the time to 10 hours, from the standard 2,970 seconds, we see an improvement in the average solution quality for these instances of about 1.51%. The average gap here for the ALNS using shorter run times is around 2.74%, which is significantly higher than what we see in Figure 5. This is due to the fact that we are dividing by the best value found using one hour, instead of as previously, using 297 seconds. The average difference between the best solutions found, between using the longer and the shorter run times, in this case, turns out to be around 0.83% and 0.97% for when using the best configurations for short and long run times, respectively. Lastly, while not better for every instance, the improved configuration for longer run times, with the 2-regret option, performs better for most of the instances.

Table 11: *Average Optimality Gap [%] for the* sensitivity instances *obtained using CPLEX.*

Optimality Gap [%]		**TW** [%]		
		30%	**15%**	**0%**
PC [%]	**20%**	21.427	25.565	31.646
	10%	24.645	29.510	35.927
	0%	27.611	32.104	39.815

Table 12: *Ratio between the best-known solution (left column) and the average solution (right column) obtained from the ALNS heuristic and the upper bound obtained using CPLEX for the sensitivity instances.*

Ratio		**TW** [%]					
		30%		**15%**		**0%**	
PC [%]	**20%**	0.989	1.001	0.985	0.999	0.975	0.991
	10%	0.983	1.001	0.971	0.992	0.972	0.991
	0%	0.978	0.994	0.971	0.988	0.958	0.978

5.4 Sensitivity Analysis

In this section, we present a study of how the difficulty of solving the PSP varies, when changing the time windows and precedence constraints for the operations. For this purpose 9 scenarios were generated, each corresponding to scheduling the operations of 12 vessels in a port with 4 terminals. Each scenario consists of a number of instances, which are identical except for that the number of precedence constraints and active time window constraints differ. Together, those instances make up the *sensitivity instances*. The number of precedence constraints considered only accounts for precedence relations between operations performed by the same vessel at two different terminals or between operations carried out at the same terminal by two different vessels. The requirement that containers from a vessel are discharged at a terminal before loading containers from the same terminal, is kept for all instances but does not count towards the number of precedence constraints in the instances. Note that the *PortLib* instances have been constructed so each instance roughly accounts that 15% of the operations are subject to strict time windows, and that the number of precedence requirements correspond to 10% of the total number of operations.

Having less constraints means that the operations can be scheduled more freely within the port, making it easier to find feasible solutions. However, this comes at the price of having a larger solution space. The aim of this study was to analyse how this trade-off affects the two solution methods.

The *sensitivity instances* were first solved using CPLEX, and the average optimality gap when varying the level of both constraints is presented in Table 11. The column **TW** indicates the percentage of operations with strict time windows, whereas the rows **PC** indicates the number of precedence requirements between operations, given as a percentage of the total number of operations. The optimality gap for the unconstrained case is relatively large, barely getting below 40% after reaching the time limit. However, as the instances become more constrained, the solution space shrinks, and the gap becomes significantly smaller.

Table 12 presents the ratios of the two solution methods. The best-known solution (to the left) and to the average solution (to the right) of the heuristic are compared to the solution found by CPLEX

for the sensitivity instances. A value below 1 means that ALNS was able to find a better solution (left column) or a better average solution (right column) than the upper bound from CPLEX. In general, the ALNS heuristic finds better solutions, and we see that the ratio between the best solution by the heuristic and by CPLEX is always below 1. We also see that the difference in performance is larger for the unconstrained case, where the heuristic found solutions that in average were around 4% better than those of CPLEX. As the problem becomes more constrained, however, the ratio between the solutions increases. This increase in ratio may also be due to the fact that the gaps for CPLEX become smaller. If parts of the improved gap comes from better integer solutions found by CPLEX, this would lower the ratio, which makes it difficult to say, whether the more constrained instances are in fact easier or more difficult to solve for the heuristic.

6 Conclusion

In this paper, we have presented a new scheduling problem for feeder vessels, which has been defined in close collaboration with the industry. The PSP accounts for all of the most important practical restrictions faced by the carriers in scheduling the operations, and a prototype is currently being tested in practice by the shipping company with which we collaborate. The developed heuristic in this work can be used as a decision support tool for planners.

We have proposed a compact formulation for the problem, inspired by machine-scheduling formulations. As the PSP has not been studied previously, we have created a set of benchmark instances, denoted *PortLib*, aiming at accurately reflecting reality. Additionally, we have shown that the PSP is $\mathcal{NP}$-hard, and only small instances could be solved to optimality by CPLEX, within the given time frame. Instead, in order to solve larger instances, an ALNS heuristic was proposed.

An extensive parameter analysis process was carried out, during which we have followed an iterative strategy to find a good parameter setting. Moreover, as part of this analysis, we studied the performance of the heuristic with different configurations of repair and destroy methods, which have proven efficient in a variety of other ALNS applications.

The result was that the combination of repair methods based on the same greedy insertion with different sorting strategies of operations performed the best when using short run times. The more elaborate destroy and repair methods, however, which uses more time per iteration, were shown to perform better than the base configuration for longer run times. Especially, the 2-regret neighbourhood increased the performance of the heuristic significantly.

The developed method showed promising results for solving the PSP. As there is no benchmark method to compare to, we compared the ALNS heuristic to CPLEX and studied the variation of the obtained results. In general it can be said that the heuristic outperforms CPLEX in nearly all cases, even when using significantly shorter run times. On 300 instances of different sizes, the average objective value of solutions found by the heuristic was better than the solution found by CPLEX for 140 of them and worse for 81, when giving the solver 10 times the run time of the heuristic. The 79 instances where the results were the same, were mainly instances where both methods found the optimal solution. In general, the heuristic performed better on larger instances, where it was also

given more time. When increasing the run time to one hour for the heuristic and 10 hours for the solver, the heuristic performed significantly better, on average, for 19 out of 20 instances.

The results further showed that longer run times have a notable positive impact on the quality of the method, both in terms of solution quality and robustness. In application, this means that the run time, and consequently the neighbourhoods, should be adjusted based on the needs of planners.

To study the effects of the constraints, we studied a variety of scenarios with an increasing number of constraints. It was clear that CPLEX benefited greatly from a more constrained search space, which was seen as a significantly lower final optimality gap. For the heuristic it is hard to compare the performance over different settings, but we could see that it performs better, in comparison to CPLEX, for less constrained instances.

For future work, it would be interesting to study the dynamic version of this problem, where the decision-making of planners is done iteratively, minimising the deviation between the operation schedule before and after the introduction of a new event. Furthermore, some terminals have sometimes enough capacity to serve more than one vessel simultaneously, so called dual-berth, but normally this comes at an additional price. If planners know which vessels can be served concurrently at the terminals, the heuristic can model it by adding proxy terminals for such operations. Otherwise, if the terminals only can operate more than one vessel at a time during certain time periods, the decision-making becomes more complex, and some of the basic structure utilised by the heuristic is changed. But it adds some additional flexibility to the model and could be an interesting subject to future work.

It is also noteworthy that the ALNS heuristic handles nonlinear objective functions, at virtually no extra cost. This makes the method significantly more flexible than working with the MIP model, and it makes it easy to adapt to the carriers needs. For future work, it would be especially interesting to see how the results would differ if the cubic nature of the fuel consumption was included in the objective function.

Lastly, in addition to being relevant in practice, the formulation turns out to be a generalisation of the classic General Shop Problem, with the possibility to model a variety of additional constraints which commonly appears in practice. We hence believe that the developed heuristic would be a good choice for various other scheduling applications.

Acknowledgements

The authors would like to thank the Danish Maritime Fund for supporting this work. The authors would also like to thank Unifeeder for valuable insight in the feeder line industry, and access to data for creating the *PortLib* instances.

References

Abreu, L. R., Cunha, J. O., Prata, B. A., and Framinan, J. M. (2020). A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Computers & Operations Research*, 113:104793.

Anand, E. and Panneerselvam, R. (2015). Literature review of open shop scheduling problems. *Intelligent Information Management*, 7(01):33.

Avriel, M. and Penn, M. (1993). Exact and approximate solutions of the container ship stowage problem. *Computers & industrial engineering*, 25(1-4):271–274.

Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., and Weglarz, J. (2019). *Handbook on scheduling*. Springer.

Brouer, B. D., Alvarez, J. F., Plum, C. E., Pisinger, D., and Sigurd, M. M. (2013). A base integer programming model and benchmark suite for liner-shipping network design. *Transportation Science*, 48(2):281–312.

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

Christiansen, M., Fagerholt, K., Nygreen, B., and Ronen, D. (2013). Ship routing and scheduling in the new millennium. *European Journal of Operational Research*, 228(3):467–483.

Christiansen, M., Hellsten, E., Pisinger, D., Sacramento, D., and Vilhelmsen, C. (2019). Liner shipping network design. *European Journal of Operational Research*. In press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Daganzo, C. F. (1989). The crane scheduling problem. *Transportation Research Part B: Methodological*, 23(3):159–175.

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). Constraint propagation techniques for the disjunctive scheduling problem. *Artificial intelligence*, 122(1-2):189–240.

Gendreau, M. and Potvin, J.-Y. (2019). Tabu search. In *Handbook of Metaheuristics*, pages 37–55. Springer.

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.

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.

Harmanani, H. M. and Ghosn, S. B. (2016). An efficient method for the open-shop scheduling problem using simulated annealing. In *Information technology: New generations*, pages 1183–1193. Springer.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 207(1):1–14.

Hosseinabadi, A. A. R., Vahidi, J., Saemi, B., Sangaiah, A. K., and Elhoseny, M. (2018). Extended genetic algorithm for solving open-shop scheduling problem. *Soft Computing*, pages 1–18.

Imai, A., Chen, H. C., Nishimura, E., and Papadimitriou, S. (2008). The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920.

Iris, Ç., Pacino, D., and Ropke, S. (2017). Improved formulations and an adaptive large neighborhood search heuristic for the integrated berth allocation and quay crane assignment problem. *Transportation Research Part E: Logistics and Transportation Review*, 105:123–147.

Kan, A. R. (2012). *Machine scheduling problems: classification, complexity and computations*. Springer Science & Business Media.

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

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220 4598:671–80.

Kokosiński, Z. and Studzienny, Ł. (2007). Hybrid genetic algorithms for the open-shop scheduling problem. *IJCSNS*, 7(9):136.

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.

Laborie, P., Rogerie, J., Shaw, P., and Vilím, P. (2018). Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250.

Liaw, C.-F. (2000). A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1):28–42.

Lin, H.-H. and Sha, D. (2011). A new particle swarm optimization for multi-objective open shop scheduling. *Journal of Information and Optimization Sciences*, 32(6):1269–1287.

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.

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

Mejía, G. and Yuraszeck, F. (2020). A self-tuning variable neighborhood search algorithm and an effective decoding scheme for open shop scheduling problems with travel/setup times. *European Journal of Operational Research*.

Muller, L. F. (2009). An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In *MIC 2009: The VIII Metaheuristics international conference*.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435.

Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.

Port of Rotterdam (2017). Samenwerken aan de haven van morgen. make it happen. Technical report, Port of Rotterdam.

Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340.

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.

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.

Sha, D. and Hsu, C.-Y. (2008). A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, 35(10):3243–3261.

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.

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

Trick, M. A. (1992). A linear relaxation heuristic for the generalized assignment problem. *Naval Research Logistics (NRL)*, 39(2):137–151.

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

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

Wang, X., Arnesen, M. J., Fagerholt, K., Gjestvang, M., and Thun, K. (2018). A two-phase heuristic for an in-port ship routing problem with tank allocation. *Computers & Operations Research*, 91:37–47.

Zhuang, Z., Huang, Z., Chen, L., and Qin, W. (2019). A heuristic rule based on complex network for open shop scheduling problem with sequence-dependent setup times and delivery times. *IEEE Access*, 7:140946–140956.

A Table of Notation

Table 13: *General Notation for the Mathematical Formulation and the ALNS heuristic.*

	Sets
$\tilde{T} = \{1, \ldots, n\} :=$	Set of Terminals.
$T = \tilde{T} \cup \{0, n+1\} :=$	Set of Terminals, including the entry and exit point of the port.
$\tilde{V} = \{1, \ldots, m\} :=$	Set of Vessels.
$V = \tilde{V} \cup \{0, m+1\} :=$	Set of Vessels, including the dummy vessels.
$O :=$	Set of operations.
$\tilde{O} :=$	Set of interior operations.
$O_i^v :=$	Set of operations to be performed by vessel $v \in V$ at terminal $i \in T$.
$O^v :=$	Set of operations for vessel $v \in V$.
$O_i :=$	Set of operations for terminal $i \in T$.
$S_i :=$	Set of closing periods for terminal $i \in T$.
$\hat{O} :=$	Set of removed operations.
	Parameters
$\delta_{ij} :=$	Sailing distance between terminal $i \in T$ and terminal $j \in T$.
$w_p :=$	Number of containers to be handled at operation $p \in O$.
$\tau_p :=$	Required time to perform operation $p \in O$.
$\lambda_{pq} :=$	Binary precedence parameter. Equal to 1 if operations $p \in O$ has to be performed after operation $q \in O$, and 0 otherwise.
$\alpha_p :=$	Time window start for operation $p \in O$.
$\beta_p :=$	Time window end for operation $p \in O$.
$\phi_p :=$	Terminal associated to operation $p \in O$.
$\nu_p :=$	Vessel associated to operation $p \in O$.
$c_p :=$	Cost coefficient of operation $p \in O$.
$Q_v :=$	Maximum cargo capacity of vessel $v \in V$.
$\hat{q}_v :=$	Initial cargo capacity of vessel $v \in V$ when arriving to the port.
$\gamma_v :=$	Priority factor for vessel $v \in V$.
$\rho :=$	Penalty weight for the departure of the vessel from the port.
$\xi_s :=$	Start time of the closing period $s \in S_i$ for terminal $i \in T$.
$\zeta_s :=$	End time of the closing period $s \in S_i$ for terminal $i \in T$.
$M_{pq} :=$	Maximum difference between the start time for operations $p, q \in O$.
$\hat{M}_{ps} :=$	Maximum difference between the start time of operation $p \in \tilde{O}$ and start of the closing period $s \in S_{\phi_p}$.
$\bar{M}_{ps} :=$	Maximum difference between the start time of operation $p \in \tilde{O}$ and the end of closing period $s \in S_{\phi_p}$.
$B :=$	Maximum operations to remove in the random removal destroy method.
$T_{st} :=$	Initial temperature factor.
$T_f :=$	End temperature factor.
$\theta :=$	Maximum allowed number of iterations without finding a new best solution.
$\pi^+ :=$	Penalty factor for updating a weight after finding a better solution.
$\pi^- :=$	Penalty factor for updating a weight after finding a worse solution.
	Decision variables
$y_p \in \mathbb{R}^+ :=$	Start time of operation $p \in O$.
$x_{pq} \in \{0, 1\} :=$	Equal to 1 if operation $p \in O$ precedes operation $q \in (O_{\phi_p} \cup O_{\nu_p}) \setminus \{p\}$.
$z_{ps} \in \{0, 1\} :=$	Equal to 1 if operation $p \in \tilde{O}$ is performed before closing period $s \in S_{\phi_p}$.

Summary of computational results for the PortLib instances

In Table 1 we see the detailed results for the *PortLib* instances with the various methods presented in *An Adaptive Large Neighbourhood Search Heuristic for Routing and Scheduling Feeder Vessels in Multi-terminal Ports*. The first column indicates the name of the name of the instance, whereas the remaining columns collect information about the CPLEX and ALNS results. From the second to sixth columns, the table reports whether CPLEX proves optimality or not, the total execution time, the percentage gap and the upper and lower bound, respectively. Furthermore, the results for the ALNS heuristics are collected in the remaining columns. The table reports the best found solution together with the average performance for the two different run times. The precise short run times are shown in Table 2 from the manuscript, whereas we set one hour execution time for every instance for the long run times. The ALNS results are from running with the best parameter settings found, as described in Table 6 in the manuscript. For the long run times, we additionally use the repair method R3, as discussed in Section 5.3.1.

Table 1: *Summary of the results for the* PortLib *instances for CPLEX and ALNS heuristic.*

Name	Status	CPLEX Time	Gap (%)	CPLEX UB	CPLEX LB	ALNS Best (Short)	ALNS Avg. (Short)	ALNS Best (Long)	ALNS Avg. (Long)	CPLEX UB (Long)
PSP.2.4.1	Optimal	0.0	0.0	15,293	15,293.0	15,293	15,293.0	-	-	-
PSP.2.4.2	Optimal	0.0	0.0	15,914	15,914.0	15,914	15,914.0	-	-	-
PSP.2.4.3	Optimal	0.0	0.0	20,883	20,883.0	20,883	20,883.0	-	-	-
PSP.2.4.4	Optimal	0.0	0.0	18,465	18,465.0	18,465	18,465.0	-	-	-
PSP.2.4.5	Optimal	0.0	0.0	14,802	14,802.0	14,802	14,802.0	-	-	-
PSP.2.4.6	Optimal	0.0	0.0	21,242	21,242.0	21,242	21,242.0	-	-	-
PSP.2.4.7	Optimal	0.0	0.0	23,589	23,589.0	23,589	23,589.0	-	-	-
PSP.2.4.8	Optimal	0.0	0.0	18,613	18,613.0	18,613	18,613.0	-	-	-
PSP.2.4.9	Optimal	0.0	0.0	9,341	9,341.0	9,341	9,341.0	-	-	-
PSP.2.4.10	Optimal	0.0	0.0	22,089	22,089.0	22,089	22,089.0	-	-	-
PSP.2.4.11	Optimal	0.0	0.0	19,852	19,852.0	19,852	19,852.0	-	-	-
PSP.2.4.12	Optimal	0.0	0.0	14,177	14,177.0	14,177	14,177.0	-	-	-
PSP.2.4.13	Optimal	0.0	0.0	16,378	16,378.0	16,378	16,378.0	-	-	-
PSP.2.4.14	Optimal	0.0	0.0	16,904	16,904.0	16,904	16,904.0	-	-	-
PSP.2.4.15	Optimal	0.0	0.0	23,084	23,084.0	23,084	23,084.0	-	-	-
PSP.2.4.16	Optimal	0.0	0.0	19,962	19,962.0	19,962	19,962.0	-	-	-
PSP.2.4.17	Optimal	0.0	0.0	28,202	28,202.0	28,202	28,202.0	-	-	-
PSP.2.4.18	Optimal	0.0	0.0	16,813	16,813.0	16,813	16,813.0	-	-	-
PSP.2.4.19	Optimal	0.0	0.0	9,353	9,353.0	9,353	9,353.0	-	-	-
PSP.2.4.20	Optimal	0.0	0.0	27,858	27,858.0	27,858	27,858.0	-	-	-
PSP.2.6.1	Optimal	0.2	0.0	26,683	26,683.0	26,683	26,683.0	-	-	-
PSP.2.6.2	Optimal	0.0	0.0	29,538	29,538.0	29,538	29,538.0	-	-	-
PSP.2.6.3	Optimal	0.1	0.0	25,984	25,982.0	25,984	25,984.0	-	-	-
PSP.2.6.4	Optimal	0.1	0.0	31,489	31,489.0	31,489	31,735.8	-	-	-
PSP.2.6.5	Optimal	0.1	0.0	32,310	32,310.0	32,310	32,310.0	-	-	-
PSP.2.6.6	Optimal	0.1	0.0	33,701	33,701.0	33,701	33,701.0	-	-	-
PSP.2.6.7	Optimal	0.0	0.0	30,582	30,582.0	30,582	30,582.0	-	-	-
PSP.2.6.8	Optimal	0.0	0.0	26,231	26,231.0	26,231	26,231.0	-	-	-
PSP.2.6.9	Optimal	0.0	0.0	35,520	35,520.0	35,520	35,520.0	-	-	-
PSP.2.6.10	Optimal	0.1	0.0	33,332	33,332.0	33,332	33,388.4	-	-	-
PSP.2.6.11	Optimal	0.0	0.0	32,951	32,951.0	32,951	32,951.0	-	-	-
PSP.2.6.12	Optimal	0.1	0.0	32,416	32,416.0	32,416	32,416.0	-	-	-
PSP.2.6.13	Optimal	0.0	0.0	47,491	47,491.0	47,491	47,491.0	-	-	-
PSP.2.6.14	Optimal	0.1	0.0	22,911	22,911.0	22,911	22,911.0	-	-	-
PSP.2.6.15	Optimal	0.3	0.0	42,867	42,866.0	42,867	42,867.0	-	-	-
PSP.2.6.16	Optimal	0.5	0.0	35,907	35,907.0	35,907	35,923.4	-	-	-
PSP.2.6.17	Optimal	0.2	0.0	35,957	35,957.0	35,957	35,957.0	-	-	-
PSP.2.6.18	Optimal	0.4	0.0	35,876	35,876.0	35,876	35,876.0	-	-	-
PSP.2.6.19	Optimal	0.0	0.0	31,752	31,752.0	31,752	31,752.0	-	-	-
PSP.2.6.20	Optimal	0.1	0.0	22,373	22,373.0	22,373	22,373.0	-	-	-
PSP.2.8.1	Optimal	0.4	0.0	57,553	57,551.8	57,553	57,553.0	-	-	-
PSP.2.8.2	Optimal	0.4	0.0	50,470	50,470.0	50,470	50,703.4	-	-	-
PSP.2.8.3	Optimal	1.6	0.0	53,920	53,920.0	53,920	53,920.0	-	-	-
PSP.2.8.4	Optimal	4.0	0.0	61,048	61,048.0	61,048	61,048.0	-	-	-
PSP.2.8.5	Optimal	3.6	0.0	74,692	74,686.3	74,692	74,838.3	-	-	-
PSP.2.8.6	Optimal	0.5	0.0	41,123	41,119.8	41,123	41,123.0	-	-	-
PSP.2.8.7	Optimal	0.4	0.0	44,182	44,182.0	44,182	44,182.0	-	-	-
PSP.2.8.8	Optimal	0.2	0.0	34,466	34,466.0	34,466	34,466.0	-	-	-
PSP.2.8.9	Optimal	0.1	0.0	61,965	61,965.0	61,965	61,965.0	-	-	-
PSP.2.8.10	Optimal	0.9	0.0	54,165	54,165.0	54,165	54,165.0	-	-	-
PSP.2.8.11	Optimal	2.9	0.0	52,584	52,579.0	52,584	52,584.0	-	-	-
PSP.2.8.12	Optimal	1.3	0.0	33,077	33,077.0	33,077	33,096.4	-	-	-
PSP.2.8.13	Optimal	2.4	0.0	56,497	56,494.7	56,497	56,497.0	-	-	-
PSP.2.8.14	Optimal	0.4	0.0	50,030	50,028.0	50,030	50,030.0	-	-	-
PSP.2.8.15	Optimal	1.0	0.0	39,822	39,822.0	39,822	39,822.0	-	-	-
PSP.2.8.16	Optimal	0.3	0.0	48,317	48,317.0	48,317	48,588.9	-	-	-
PSP.2.8.17	Optimal	5.2	0.0	53,957	53,954.0	53,957	53,957.0	-	-	-
PSP.2.8.18	Optimal	0.8	0.0	57,470	57,468.0	57,470	57,470.0	-	-	-
PSP.2.8.19	Optimal	1.5	0.0	59,245	59,243.0	59,245	59,245.0	-	-	-

PSP.2.8.20	Optimal	0.6	0.0	50,094	50,090.0	50,094	50,094.0	-	-	-
PSP.3.6.1	Optimal	1.3	0.0	42,381	42,381.0	42,381	42,507.0	-	-	-
PSP.3.6.2	Optimal	0.5	0.0	43,977	43,977.0	43,977	43,977.0	-	-	-
PSP.3.6.3	Optimal	1.3	0.0	49,286	49,286.0	49,286	49,294.1	-	-	-
PSP.3.6.4	Optimal	2.5	0.0	41,742	41,742.0	41,742	42,060.0	-	-	-
PSP.3.6.5	Optimal	0.5	0.0	54,388	54,382.6	54,388	54,388.0	-	-	-
PSP.3.6.6	Optimal	0.8	0.0	34,666	34,666.0	34,666	34,666.0	-	-	-
PSP.3.6.7	Optimal	1.8	0.0	51,304	51,304.0	51,304	51,337.2	-	-	-
PSP.3.6.8	Optimal	1.7	0.0	30,307	30,307.0	30,307	30,310.3	-	-	-
PSP.3.6.9	Optimal	5.3	0.0	31,941	31,941.0	31,941	31,948.2	-	-	-
PSP.3.6.10	Optimal	0.4	0.0	44,923	44,921.3	44,923	44,923.0	-	-	-
PSP.3.6.11	Optimal	4.9	0.0	43,550	43,548.7	43,550	43,550.0	-	-	-
PSP.3.6.12	Optimal	2.6	0.0	36,925	36,925.0	36,925	36,925.0	-	-	-
PSP.3.6.13	Optimal	1.7	0.0	49,662	49,662.0	49,662	49,662.0	-	-	-
PSP.3.6.14	Optimal	16.4	0.0	49,529	49,524.1	49,529	49,529.0	-	-	-
PSP.3.6.15	Optimal	1.3	0.0	43,094	43,094.0	43,094	43,094.0	-	-	-
PSP.3.6.16	Optimal	0.9	0.0	38,691	38,689.6	38,691	38,691.0	-	-	-
PSP.3.6.17	Optimal	0.4	0.0	44,672	44,670.3	44,672	44,672.0	-	-	-
PSP.3.6.18	Optimal	2.2	0.0	47,424	47,422.7	47,424	47,424.0	-	-	-
PSP.3.6.19	Optimal	1.4	0.0	58,768	58,764.3	58,768	58,875.8	-	-	-
PSP.3.6.20	Optimal	4.0	0.0	49,246	49,244.1	49,246	49,290.8	-	-	-
PSP.3.8.1	Optimal	11.7	0.0	66,345	66,340.0	66,345	66,345.0	-	-	-
PSP.3.8.2	Optimal	37.2	0.0	69,911	69,904.6	69,911	69,911.0	-	-	-
PSP.3.8.3	Optimal	11.1	0.0	58,026	58,020.3	58,026	58,046.0	-	-	-
PSP.3.8.4	Optimal	191.1	0.0	76,588	76,580.5	76,588	76,588.0	-	-	-
PSP.3.8.5	Feasible	200	9.7	68,696	62,038.3	68,541	68,541.0	-	-	-
PSP.3.8.6	Optimal	4.0	0.0	60,030	60,027.4	60,030	60,030.0	-	-	-
PSP.3.8.7	Feasible	200	11.5	78,894	69,806.5	78,744	78,753.0	-	-	-
PSP.3.8.8	Optimal	108.5	0.0	62,413	62,406.8	62,413	62,413.0	-	-	-
PSP.3.8.9	Optimal	10.1	0.0	67,947	67,943.7	67,947	68,099.7	-	-	-
PSP.3.8.10	Optimal	16.0	0.0	71,038	71,032.0	71,038	71,038.0	-	-	-
PSP.3.8.11	Optimal	35.7	0.0	67,673	67,666.2	67,673	67,673.0	-	-	-
PSP.3.8.12	Optimal	1.3	0.0	106,042	106,042.0	106,042	106,042.0	-	-	-
PSP.3.8.13	Optimal	6.6	0.0	47,129	47,129.0	47,129	47,188.8	-	-	-
PSP.3.8.14	Optimal	68.6	0.0	54,496	54,496.0	54,496	54,496.0	-	-	-
PSP.3.8.15	Optimal	17.1	0.0	68,643	68,643.0	68,643	68,643.0	-	-	-
PSP.3.8.16	Optimal	3.8	0.0	59,586	59,581.7	59,586	59,586.0	-	-	-
PSP.3.8.17	Optimal	56.6	0.0	59,868	59,862.7	59,992	59,946.6	-	-	-
PSP.3.8.18	Feasible	200	14.8	80,820	68,877.3	80,820	80,847.9	-	-	-
PSP.3.8.19	Optimal	4.7	0.0	55,230	55,224.9	55,230	55,230.0	-	-	-
PSP.3.8.20	Optimal	109.8	0.0	66,857	66,850.3	66,857	67,020.1	-	-	-
PSP.3.10.1	Optimal	11.1	0.0	75,273	75,265.8	75,273	75,475.6	-	-	-
PSP.3.10.2	Feasible	380	22.8	109,196	84,261.1	107,569	108,140.3	-	-	-
PSP.3.10.3	Optimal	171.5	0.0	102,779	102,768.7	102,779	102,790.6	-	-	-
PSP.3.10.4	Feasible	380	18.3	74,684	61,006.0	74,092	74,092.0	-	-	-
PSP.3.10.5	Optimal	116.8	0.0	96,243	96,233.4	96,243	96,243.0	-	-	-
PSP.3.10.6	Feasible	380	5.3	91,589	86,737.1	91,589	91,750.4	-	-	-
PSP.3.10.7	Feasible	380	14.4	93,522	80,015.9	93,522	93,536.5	-	-	-
PSP.3.10.8	Feasible	380	10.6	75,989	67,945.1	75,989	75,997.6	-	-	-
PSP.3.10.9	Optimal	80.0	0.0	115,502	115,490.9	115,502	115,520.2	-	-	-
PSP.3.10.10	Optimal	267.3	0.0	90,338	90,329.2	91,008	90,823.8	-	-	-
PSP.3.10.11	Feasible	380	5.8	62,068	58,445.8	62,068	62,227.5	-	-	-
PSP.3.10.12	Feasible	380	6.4	84,418	79,021.7	84,400	84,400.0	-	-	-
PSP.3.10.13	Optimal	33.6	0.0	85,772	85,764.1	85,772	85,968.4	-	-	-
PSP.3.10.14	Feasible	380	1.7	109,935	108,065.4	109,935	109,951.6	-	-	-
PSP.3.10.15	Feasible	380	30.8	126,336	87,390.9	126,056	126,400.3	-	-	-
PSP.3.10.16	Optimal	99.3	0.0	112,178	112,167.1	112,178	112,458.1	-	-	-
PSP.3.10.17	Feasible	380	15.7	91,676	77,261.9	90,585	90,785.9	-	-	-
PSP.3.10.18	Feasible	380	14.5	87,249	74,620.7	86,082	86,223.8	-	-	-
PSP.3.10.19	Feasible	380	27.0	93,946	68,566.9	93,946	94,065.5	-	-	-
PSP.3.10.20	Feasible	380	15.6	101,921	85,975.9	101,578	101,578.0	-	-	-
PSP.3.12.1	Feasible	670	35.3	136,930	88,632.8	134,183	135,134.6	-	-	-
PSP.3.12.2	Optimal	300.7	0.0	143,468	143,454.1	143,468	143,468.0	-	-	-
PSP.3.12.3	Feasible	670	33.0	143,368	96,005.2	139,834	140,033.0	-	-	-
PSP.3.12.4	Feasible	670	24.8	119,890	90,113.6	116,827	117,374.8	-	-	-
PSP.3.12.5	Feasible	670	23.9	103,011	78,431.4	102,675	102,538.2	-	-	-
PSP.3.12.6	Feasible	670	18.4	127,060	103,666.9	124,417	124,434.3	-	-	-
PSP.3.12.7	Feasible	670	18.1	124,438	101,918.4	124,438	130,010.1	-	-	-
PSP.3.12.8	Feasible	670	31.0	130,030	89,700.7	130,030	129,708.1	-	-	-
PSP.3.12.9	Feasible	670	18.9	104,190	84,447.8	103,362	103,407.2	-	-	-
PSP.3.12.10	Feasible	670	11.6	108,457	95,842.9	108,450	108,450.0	-	-	-
PSP.3.12.11	Feasible	670	21.7	126,992	99,449.9	124,843	125,739.1	-	-	-
PSP.3.12.12	Feasible	670	25.9	105,008	77,855.5	102,658	102,903.7	-	-	-
PSP.3.12.13	Feasible	670	24.1	160,584	121,844.9	157,189	159,257.1	-	-	-
PSP.3.12.14	Feasible	670	17.0	124,035	102,929.9	122,731	122,797.2	-	-	-
PSP.3.12.15	Feasible	670	30.2	103,755	72,443.3	102,617	103,002.2	-	-	-
PSP.3.12.16	Feasible	670	20.4	89,073	70,926.1	88,009	89,886.8	-	-	-
PSP.3.12.17	Feasible	670	24.3	156,999	118,915.2	155,118	156,358.8	-	-	-
PSP.3.12.18	Feasible	670	19.4	85,174	68,686.3	85,174	85,930.9	-	-	-
PSP.3.12.19	Feasible	670	9.8	123,080	110,987.8	123,080	123,331.6	-	-	-
PSP.3.12.20	Feasible	670	1.9	122,645	120,302.6	122,659	122,671.7	-	-	-

PSP.4.8.1	Feasible	460	22.5	71,901	55,711.0	71,765	72,178.7	-	-	-
PSP.4.8.2	Optimal	87.3	0.0	93,767	93,758.0	93,767	93,998.3	-	-	-
PSP.4.8.3	Optimal	195.4	0.0	61,997	61,990.8	62,523	62,391.4	-	-	-
PSP.4.8.4	Feasible	460	22.5	59,401	46,029.5	58,714	58,642.4	-	-	-
PSP.4.8.5	Feasible	460	13.3	76,534	66,381.0	75,017	75,767.9	-	-	-
PSP.4.8.6	Feasible	460	15.3	77,543	65,704.5	76,681	76,762.3	-	-	-
PSP.4.8.7	Feasible	460	7.0	79,008	73,459.0	78,491	78,945.3	-	-	-
PSP.4.8.8	Feasible	460	5.2	72,047	68,290.5	72,380	72,444.0	-	-	-
PSP.4.8.9	Feasible	460	8.6	91,350	83,466.7	91,583	91,507.3	-	-	-
PSP.4.8.10	Feasible	460	12.9	68,984	60,092.3	68,869	68,999.1	-	-	-
PSP.4.8.11	Feasible	460	4.6	76,339	72,791.0	76,339	76,601.5	-	-	-
PSP.4.8.12	Optimal	273.8	0.0	67,608	67,601.3	67,916	68,290.7	-	-	-
PSP.4.8.13	Feasible	460	14.3	87,801	75,214.7	87,542	88,254.9	-	-	-
PSP.4.8.14	Feasible	460	12.5	86,981	76,139.6	85,058	85,631.4	-	-	-
PSP.4.8.15	Feasible	460	13.6	82,314	71,111.3	81,668	83,026.5	-	-	-
PSP.4.8.16	Optimal	179.7	0.0	79,575	79,567.1	79,647	79,596.6	-	-	-
PSP.4.8.17	Feasible	460	12.2	75,525	66,310.5	74,594	74,924.8	-	-	-
PSP.4.8.18	Feasible	460	7.5	63,037	58,309.3	62,977	63,032.5	-	-	-
PSP.4.8.19	Feasible	460	16.6	90,438	75,396.3	90,379	90,736.9	-	-	-
PSP.4.8.20	Feasible	460	15.2	83,282	70,660.0	81,759	81,955.4	-	-	-
PSP.4.10.1	Feasible	880	14.2	99,762	85,591.9	98,941	99,239.9	-	-	-
PSP.4.10.2	Feasible	880	23.5	97,145	74,292.1	96,161	96,549.2	-	-	-
PSP.4.10.3	Feasible	880	15.9	115,681	97,262.1	115,386	115,249.8	-	-	-
PSP.4.10.4	Feasible	880	8.9	125,715	114,487.9	125,715	125,898.3	-	-	-
PSP.4.10.5	Feasible	880	16.8	146,195	121,645.6	147,690	147,590.7	-	-	-
PSP.4.10.6	Feasible	880	13.2	85,551	74,223.1	84,525	84,732.3	-	-	-
PSP.4.10.7	Feasible	880	31.7	90,123	61,560.1	87,753	88,819.3	-	-	-
PSP.4.10.8	Feasible	880	25.4	115,244	85,968.7	112,634	113,114.0	-	-	-
PSP.4.10.9	Feasible	880	23.0	122,427	94,232.4	119,179	119,441.6	-	-	-
PSP.4.10.10	Feasible	880	35.4	150,267	97,111.8	142,853	144,731.9	-	-	-
PSP.4.10.11	Feasible	880	24.1	101,989	77,372.4	100,412	100,420.9	-	-	-
PSP.4.10.12	Feasible	880	24.7	116,686	87,810.3	116,535	117,237.3	-	-	-
PSP.4.10.13	Feasible	880	21.3	133,906	105,423.8	130,727	131,627.3	-	-	-
PSP.4.10.14	Feasible	880	35.0	80,570	52,390.0	77,697	77,720.5	-	-	-
PSP.4.10.15	Feasible	880	24.6	134,574	101,509.7	132,872	133,077.6	-	-	-
PSP.4.10.16	Feasible	880	39.5	96,034	58,122.6	90,911	92,653.0	-	-	-
PSP.4.10.17	Feasible	880	25.0	107,177	80,368.3	104,168	105,618.9	-	-	-
PSP.4.10.18	Feasible	880	20.8	110,929	87,873.4	110,167	110,761.2	-	-	-
PSP.4.10.19	Feasible	880	35.4	127,686	82,542.2	123,702	124,537.1	-	-	-
PSP.4.10.20	Feasible	880	16.4	117,897	98,560.3	113,995	117,556.8	-	-	-
PSP.4.12.1	Feasible	1580	29.7	184,227	129,440.3	185,473	187,150.8	-	-	-
PSP.4.12.2	Feasible	1580	40.9	175,436	103,627.4	162,781	167,071.9	-	-	-
PSP.4.12.3	Feasible	1580	34.4	138,424	90,838.5	128,906	132,339.4	-	-	-
PSP.4.12.4	Feasible	1580	41.5	162,136	94,866.6	154,726	155,920.7	-	-	-
PSP.4.12.5	Feasible	1580	17.8	144,082	118,379.4	140,125	142,047.9	-	-	-
PSP.4.12.6	Feasible	1580	38.4	150,436	92,606.1	148,242	152,041.6	-	-	-
PSP.4.12.7	Feasible	1580	38.1	158,067	97,824.6	155,599	156,953.8	-	-	-
PSP.4.12.8	Feasible	1580	18.3	178,600	145,865.8	175,913	178,140.7	-	-	-
PSP.4.12.9	Feasible	1580	25.4	152,046	113,496.6	151,412	154,550.9	-	-	-
PSP.4.12.10	Feasible	1580	24.9	144,648	108,619.8	141,212	142,253.3	-	-	-
PSP.4.12.11	Feasible	1580	27.4	123,632	89,805.7	121,990	122,603.4	-	-	-
PSP.4.12.12	Feasible	1580	32.1	169,199	114,963.8	159,512	160,280.8	-	-	-
PSP.4.12.13	Feasible	1580	31.2	164,142	112,972.7	161,214	163,772.9	-	-	-
PSP.4.12.14	Feasible	1580	27.3	118,688	86,284.9	117,012	118,757.1	-	-	-
PSP.4.12.15	Feasible	1580	30.9	139,127	96,162.4	138,544	139,441.5	-	-	-
PSP.4.12.16	Feasible	1580	42.7	150,724	86,304.8	143,205	143,912.7	-	-	-
PSP.4.12.17	Feasible	1580	26.5	121,635	89,398.5	118,476	119,180.0	-	-	-
PSP.4.12.18	Feasible	1580	34.7	140,058	91,502.1	128,648	128,991.5	-	-	-
PSP.4.12.19	Feasible	1580	25.7	206,306	153,238.6	200,601	202,928.6	-	-	-
PSP.4.12.20	Feasible	1580	35.7	167,881	107,869.4	161,923	164,526.8	-	-	-
PSP.4.14.1	Feasible	2470	36.0	192,728	123,416.0	184,640	185,810.7	-	-	-
PSP.4.14.2	Feasible	2470	38.7	191,702	117,567.9	185,342	186,533.4	-	-	-
PSP.4.14.3	Feasible	2470	42.8	181,728	103,936.0	176,522	177,715.8	-	-	-
PSP.4.14.4	Feasible	2470	24.8	254,594	191,535.8	254,838	255,849.1	-	-	-
PSP.4.14.5	Feasible	2470	36.8	202,398	127,868.6	200,362	203,058.3	-	-	-
PSP.4.14.6	Feasible	2470	29.9	176,190	123,526.4	175,328	177,862.7	-	-	-
PSP.4.14.7	Feasible	2470	33.0	203,029	135,947.5	198,513	201,967.8	-	-	-
PSP.4.14.8	Feasible	2470	37.8	202,541	126,051.0	197,365	200,375.1	-	-	-
PSP.4.14.9	Feasible	2470	42.3	205,267	118,359.0	191,266	194,958.3	-	-	-
PSP.4.14.10	Feasible	2470	37.0	261,398	164,556.2	244,920	248,202.0	-	-	-
PSP.4.14.11	Feasible	2470	34.2	227,671	149,818.4	226,688	226,904.8	-	-	-
PSP.4.14.12	Feasible	2470	33.6	187,314	124,426.8	183,329	187,869.9	-	-	-
PSP.4.14.13	Feasible	2470	45.0	175,101	96,324.5	170,326	171,787.2	-	-	-
PSP.4.14.14	Feasible	2470	30.1	228,420	159,637.6	219,243	221,450.2	-	-	-
PSP.4.14.15	Feasible	2470	30.7	188,429	130,539.8	185,358	186,694.2	-	-	-
PSP.4.14.16	Feasible	2470	42.6	175,207	100,602.8	170,697	174,586.1	-	-	-
PSP.4.14.17	Feasible	2470	42.1	177,267	102,724.3	175,211	175,914.9	-	-	-
PSP.4.14.18	Feasible	2470	27.7	193,229	139,681.9	189,882	191,212.2	-	-	-
PSP.4.14.19	Feasible	2470	31.8	179,422	122,387.7	174,479	176,919.7	-	-	-
PSP.4.14.20	Feasible	2470	29.6	206,342	145,313.7	201,274	206,562.2	-	-	-
PSP.5.10.1	Feasible	1720	21.7	111,525	87,331.4	107,575	108,915.4	107,575	107,785.0	-

PSP.5.10.2	Feasible	1720	21.7	138,769	108,709.7	135,988	137,318.4	135,988	135,988.0	-
PSP.5.10.3	Feasible	1720	31.9	107,500	73,223.9	106,081	107,593.9	104,309	104,815.8	-
PSP.5.10.4	Feasible	1720	23.4	105,735	81,026.8	105,160	105,654.1	104,647	104,983.2	-
PSP.5.10.5	Feasible	1720	27.2	129,429	94,173.0	125,671	128,537.5	125,671	126,523.0	-
PSP.5.10.6	Feasible	1720	18.0	119,233	97,772.8	119,048	120,296.1	118,158	118,581.0	-
PSP.5.10.7	Feasible	1720	30.5	107,445	74,623.7	107,554	108,302.4	106,479	107,101.8	-
PSP.5.10.8	Feasible	1720	14.8	148,725	126,663.9	146,728	147,391.2	146,210	146,440.0	-
PSP.5.10.9	Feasible	1720	39.7	128,935	77,691.2	124,148	124,812.6	121,099	121,856.0	-
PSP.5.10.10	Feasible	1720	26.3	122,242	90,071.8	124,680	125,895.6	124,164	124,165.6	-
PSP.5.10.11	Feasible	1720	18.6	118,872	96,796.7	118,306	119,571.8	117,918	118,375.8	-
PSP.5.10.12	Feasible	1720	35.4	122,208	78,946.9	117,978	121,421.2	118,261	118,813.6	-
PSP.5.10.13	Feasible	1720	25.2	126,516	94,638.2	122,334	124,031.1	121,251	121,974.8	-
PSP.5.10.14	Feasible	1720	27.6	126,565	91,663.2	124,127	125,299.7	123,980	124,075.4	-
PSP.5.10.15	Feasible	1720	22.8	106,634	82,349.6	105,647	106,022.2	103,399	104,206.4	-
PSP.5.10.16	Feasible	1720	29.5	152,067	107,213.0	148,423	151,191.3	148,358	149,118.2	-
PSP.5.10.17	Feasible	1720	32.4	121,628	82,248.3	113,896	114,871.8	112,526	113,337.4	-
PSP.5.10.18	Feasible	1720	36.7	165,161	104,489.8	161,358	162,091.9	159,092	159,304.2	-
PSP.5.10.19	Feasible	1720	35.4	122,578	79,176.9	118,784	118,403.5	115,915	116,409.0	-
PSP.5.10.20	Feasible	1720	34.8	132,558	86,437.5	131,767	132,545.8	130,041	130,440.8	-
PSP.5.12.1	Feasible	2970	39.5	201,337	121,756.4	200,536	205,533.0	200,423	201,376.8	198,534
PSP.5.12.2	Feasible	2970	34.5	179,988	117,937.4	172,328	175,695.9	169,103	170,494.8	172,157
PSP.5.12.3	Feasible	2970	43.0	191,828	109,272.8	186,283	187,184.3	182,335	185,053.6	190,840
PSP.5.12.4	Feasible	2970	36.8	174,174	110,151.0	169,517	171,654.7	166,514	168,423.4	170,116
PSP.5.12.5	Feasible	2970	35.9	168,738	108,098.9	161,092	161,042.9	159,449	159,811.4	164,115
PSP.5.12.6	Feasible	2970	38.8	165,819	101,533.7	161,866	162,911.8	159,882	161,964.8	163,348
PSP.5.12.7	Feasible	2970	31.4	189,069	129,739.2	178,263	186,284.2	180,135	182,000.8	183,880
PSP.5.12.8	Feasible	2970	21.9	153,167	119,638.8	153,117	154,369.7	147,954	149,866.0	152,468
PSP.5.12.9	Feasible	2970	38.6	163,481	100,305.5	158,303	162,435.9	157,253	159,904.8	163,911
PSP.5.12.10	Feasible	2970	32.0	181,628	123,425.1	175,702	179,594.7	174,359	176,607.6	178,927
PSP.5.12.11	Feasible	2970	24.6	154,029	116,063.0	149,704	152,583.4	146,765	148,480.6	148,900
PSP.5.12.12	Feasible	2970	31.6	157,722	107,920.2	157,430	157,782.8	155,405	156,544.6	159,344
PSP.5.12.13	Feasible	2970	30.4	170,137	118,438.8	169,600	171,215.2	166,926	168,087.8	169,943
PSP.5.12.14	Feasible	2970	35.6	180,642	116,391.5	173,232	174,323.9	170,943	172,965.0	174,566
PSP.5.12.15	Feasible	2970	31.8	201,715	137,483.3	197,534	201,743.0	195,982	197,507.4	202,687
PSP.5.12.16	Feasible	2970	29.7	169,855	119,329.0	170,751	171,549.0	167,008	168,981.8	170,928
PSP.5.12.17	Feasible	2970	39.6	235,708	142,454.3	225,797	226,510.5	220,691	221,556.4	231,423
PSP.5.12.18	Feasible	2970	33.8	163,008	107,953.7	154,851	157,747.9	153,908	156,195.4	160,756
PSP.5.12.19	Feasible	2970	39.4	190,492	115,346.7	182,310	186,000.8	182,790	183,752.2	185,335
PSP.5.12.20	Feasible	2970	32.9	172,198	115,560.1	164,414	165,996.2	162,682	164,087.4	167,953
PSP.5.14.1	Feasible	4710	52.1	258,843	124,007.4	246,102	249,034.8	243,670	244,488.4	-
PSP.5.14.2	Feasible	4710	35.5	203,240	131,069.7	200,908	204,199.1	199,113	201,111.4	-
PSP.5.14.3	Feasible	4710	35.8	237,827	152,659.1	233,049	238,312.5	232,671	234,377.2	-
PSP.5.14.4	Feasible	4710	37.4	190,948	119,522.1	186,225	186,979.0	182,359	185,533.4	-
PSP.5.14.5	Feasible	4710	42.8	224,953	128,635.9	213,905	215,554.2	211,338	212,391.6	-
PSP.5.14.6	Feasible	4710	32.0	263,710	179,217.2	253,416	253,996.7	250,341	251,829.2	-
PSP.5.14.7	Feasible	4710	40.7	225,930	134,015.0	219,515	220,672.5	218,020	218,802.8	-
PSP.5.14.8	Feasible	4710	37.3	177,294	111,195.1	175,260	175,127.3	172,499	173,427.6	-
PSP.5.14.9	Feasible	4710	38.9	211,067	128,984.6	202,710	203,965.2	200,850	203,365.6	-
PSP.5.14.10	Feasible	4710	44.9	181,409	100,007.9	166,888	169,532.8	166,511	167,495.6	-
PSP.5.14.11	Feasible	4710	29.3	205,794	145,509.7	200,948	201,124.6	198,488	200,046.4	-
PSP.5.14.12	Feasible	4710	34.9	238,794	155,562.5	233,117	237,087.5	232,390	235,346.8	-
PSP.5.14.13	Feasible	4710	30.4	254,311	176,953.9	249,375	252,672.3	248,149	250,196.8	-
PSP.5.14.14	Feasible	4710	47.7	223,499	116,912.5	214,039	216,773.5	212,374	215,542.4	-
PSP.5.14.15	Feasible	4710	38.9	312,803	191,258.9	293,933	297,825.3	290,894	293,764.0	-
PSP.5.14.16	Feasible	4710	47.9	230,618	120,125.2	216,154	218,661.9	214,286	215,636.0	-
PSP.5.14.17	Feasible	4710	36.2	184,333	117,570.5	182,577	185,077.9	180,315	181,632.2	-
PSP.5.14.18	Feasible	4710	34.7	273,073	178,272.8	264,777	267,521.0	261,489	264,208.4	-
PSP.5.14.19	Feasible	4710	30.4	229,802	159,990.7	223,981	225,263.4	217,262	221,016.0	-
PSP.5.14.20	Feasible	4710	34.1	192,353	126,819.5	183,782	185,383.3	183,408	184,206.2	-
PSP.5.16.1	Feasible	7030	28.1	312,288	224,429.7	306,831	311,669.1	304,841	308,913.0	-
PSP.5.16.2	Feasible	7030	43.6	230,796	130,128.8	217,592	222,339.5	217,308	219,355.8	-
PSP.5.16.3	Feasible	7030	29.5	213,835	150,801.7	213,511	216,723.8	212,473	213,511.2	-
PSP.5.16.4	Feasible	7030	40.4	252,336	150,424.1	236,437	244,857.3	238,885	241,203.8	-
PSP.5.16.5	Feasible	7030	35.8	273,893	175,875.1	266,035	269,238.0	264,412	270,823.2	-
PSP.5.16.6	Feasible	7030	41.9	263,450	153,184.4	241,263	246,755.1	240,978	243,333.4	-
PSP.5.16.7	Feasible	7030	44.5	319,136	177,111.8	301,739	308,827.4	299,743	306,127.6	-
PSP.5.16.8	Feasible	7030	44.8	226,712	125,061.4	221,382	225,710.0	220,993	222,839.8	-
PSP.5.16.9	Feasible	7030	33.1	259,384	173,415.6	249,949	254,266.8	249,449	252,294.2	-
PSP.5.16.10	Feasible	7030	36.8	371,843	234,904.4	350,639	355,099.5	349,071	351,271.4	-
PSP.5.16.11	Feasible	7030	35.3	287,836	186,088.0	281,495	283,333.5	279,149	281,179.8	-
PSP.5.16.12	Feasible	7030	40.8	245,892	145,514.3	231,876	235,485.4	232,984	234,295.2	-
PSP.5.16.13	Feasible	7030	39.5	289,990	175,569.1	281,589	286,959.5	280,129	283,986.4	-
PSP.5.16.14	Feasible	7030	41.3	241,622	141,834.8	235,009	239,518.8	232,388	235,427.8	-
PSP.5.16.15	Feasible	7030	33.9	318,835	210,736.3	301,210	309,660.0	297,431	302,690.4	-
PSP.5.16.16	Feasible	7030	36.7	353,614	223,721.8	335,194	340,773.0	334,278	336,273.6	-
PSP.5.16.17	Feasible	7030	47.1	305,538	161,722.9	303,141	307,152.7	297,061	301,695.4	-
PSP.5.16.18	Feasible	7030	45.8	243,146	131,820.4	238,994	248,000.6	236,513	244,114.4	-
PSP.5.16.19	Feasible	7030	35.7	314,108	201,963.0	298,500	303,361.2	296,325	300,703.4	-
PSP.5.16.20	Feasible	7030	42.2	274,204	158,588.4	258,593	262,173.0	257,894	260,204.6	-