Improving Inland and Short-Sea Vessel Scheduling using Constraint Optimization

A Google OR-Tools Implementation for a Container Vessel Planning System

Master's Thesis

Daniel Chou Rainho

Delft University of Technology

TU Delft

Improving Inland and Short-Sea Vessel Scheduling using Constraint Optimization

A Google OR-Tools Implementation for a Container Vessel Planning System

by

Daniel Chou Rainho

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday April 17, 2026 at 12:00 PM.

Student number:	5319412	
Project duration:	September 8, 2025 – March 27, 2026	
Thesis committee:	Dr. N. Yorke-Smith,	TU Delft, supervisor
	Dr. B. Atasoy,	TU Delft
	Drs. Q. Schevenhoven,	QEWI

Cover: Container barge on an inland waterway by dendoktoor on Pixabay (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This thesis marks the completion of my Master's degree in Computer Science at Delft University of Technology. The work was carried out in collaboration with a Dutch logistics software company, where I had the opportunity to apply optimization techniques to a real-world vessel scheduling problem.

I would like to thank Neil Yorke-Smith, my university supervisor, for his academic guidance and feedback throughout this project. I am also grateful to Quirijn and Frank, my company supervisors, for their continuous support and domain expertise.

Daniel Chou Rainho
Delft, April 2026

Summary

Inland and short-sea container shipping in Northwestern Europe relies on manual planning by experienced logistics operators. This process, while effective for routine operations, is time-consuming, difficult to scale, and limited in its ability to globally optimize fleet utilization. The underlying scheduling problem is classified as a Heterogeneous Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW), incorporating domain-specific constraints such as minimum call sizes, forbidden terminals, terminal opening hours, and mandatory vessel breaks.

This thesis presents Orion, a constraint-based optimization solver built on the Google OR-Tools Routing Library, designed to automate and improve this vessel scheduling process. The problem is formulated using Constraint Programming, which allows complex business rules to be expressed as logical predicates rather than linearized inequalities. The solver pipeline includes a data preprocessing stage that aggregates individual container orders into compound orders, reducing problem size by approximately 90% while preserving solution quality.

The evaluation follows a three-phase methodology. First, a feasibility analysis filters 14 construction heuristics down to two viable candidates: Local Cheapest Insertion and Parallel Cheapest Insertion. Second, a parameter tuning phase identifies Local Cheapest Insertion combined with Tabu Search as the best-performing configuration, achieving an Average Relative Percentage Deviation of 2.72% across all scenarios. Third, a benchmarking phase compares Orion against human planners and the company's existing Simulated Annealing solver (Baseline SA) on six real-world scenarios from three distinct logistics operators.

The results show that Orion achieves travel cost reductions of 6.7% to 32.8% compared to human planners on five of six scenarios and outperforms Baseline SA's average result on five of six scenarios. A key structural advantage is determinism: Orion produces identical results across repeated runs, whereas Baseline SA exhibits variance of up to 37.5% between its best and worst runs. Convergence analysis indicates that 96–99.97% of objective improvement occurs within the first 30 seconds, making a 5-minute time budget sufficient for operational use.

The thesis concludes with recommendations for production deployment and identifies future research directions, including dynamic water level constraints, improved rolling-horizon replanning, custom fleet reduction operators, and hub-based consolidation strategies.

Contents

Preface i

Summary ii

Nomenclature vi

1 Introduction 1
1.1 Context and Motivation 1
1.2 Company and Problem Description 1
1.3 Research Goal and Objectives 2
1.4 Research Questions 2
1.5 Thesis Outline 2

2 Problem Context 4
2.1 Waterborne Container Transport 4
2.1.1 Current Planning Process 4
2.1.2 Operational Entities 5
2.1.3 Operational Constraints 5
2.2 The Vehicle Routing Problem 6
2.3 Classification of the Problem 6
2.4 Computational Complexity 7
2.5 Conclusion 7

3 Theoretical Framework 8
3.1 Mathematical Modelling Frameworks 8
3.1.1 Mixed Integer Linear Programming 8
3.1.2 Constraint Programming 8
3.1.3 Handling Constraints: Hard vs. Soft 9
3.2 Solution Methodologies 9
3.2.1 Exact Methods 10
3.2.2 Construction Heuristics 10
3.2.3 Improvement Heuristics: Local Search 11
3.2.4 Metaheuristics: Escaping Local Optima 13

4 Problem Formulation 16
4.1 Detailed Problem Definition 16
4.1.1 Illustrative Data Example 16
4.2 Formal Problem Description 17
4.2.1 Sets and Parameters 17
4.2.2 Decision Variables 18
4.2.3 Objective Function 18
4.2.4 Operational Constraints 19
4.3 Model Assumptions 20
4.3.1 Wild Anchorage 20
4.3.2 Deterministic Travel Times 20
4.3.3 Static Capacity 21
4.3.4 Fixed Drop Penalty 21

5 Implementation with Google OR-Tools 22
5.1 Architecture and Design Rationale 22
5.1.1 Strategic Choice: Why Google OR-Tools? 22
5.1.2 The CP-SAT Exploratory Phase 22

5.1.3 The Scalability Bottleneck . . . 23
5.2 System Architecture . . . 24
5.2.1 The Modeler Pipeline . . . 25
5.3 Data Reduction and Solver Mapping . . . 25
5.3.1 Order Aggregation Strategy . . . 25
5.3.2 Index Mapping . . . 26
5.3.3 Cost Scaling . . . 26
5.4 Implementing Core Dimensions . . . 26
5.4.1 Time and Variable Speed . . . 26
5.4.2 Capacity Constraints . . . 27
5.4.3 Charter (Day) Costs . . . 27
5.4.4 Modal Shift and Unplanned Orders . . . 27
5.5 Implementing Complex Business Rules . . . 28
5.5.1 Binding Onboard Cargo . . . 28
5.5.2 Minimum Call Size: The Penalty Gradient . . . 28
5.5.3 Minimum Overtime Call Size (MOCS) . . . 28
5.5.4 Mandatory Break Intervals . . . 29
5.5.5 Forbidden Terminals . . . 29
5.5.6 Terminal Opening Hours . . . 29
5.6 Search Configuration . . . 29
5.6.1 Automated Diagnosis Tool . . . 29
5.7 Baseline Solvers . . . 29
5.7.1 Shared Construction Heuristic . . . 30
5.7.2 Baseline SA: Simulated Annealing . . . 30
5.7.3 Baseline LNS: Destroy-and-Repair . . . 30

6 Evaluation **31**
6.1 Experimental Setup . . . 31
6.1.1 Environment . . . 31
6.1.2 Datasets . . . 31
6.1.3 Key Performance Indicators . . . 32
6.1.4 Baselines . . . 32
6.2 Phase 0: Feasibility Analysis . . . 33
6.3 Phase 1: Parameter Tuning . . . 33
6.3.1 Ranking by ARPD . . . 33
6.3.2 Observations . . . 33
6.3.3 Local Search Operator Activity . . . 34
6.3.4 Selected Configurations . . . 35
6.4 Phase 2: Benchmark Comparison . . . 35
6.4.1 Extended Orion Runs . . . 36
6.4.2 Baseline SA Results . . . 36
6.4.3 Extended Baseline SA Runs . . . 36
6.4.4 Head-to-Head Comparison . . . 37
6.4.5 Analysis per Scenario . . . 37
6.5 Discussion . . . 38
6.5.1 Comparative Analysis . . . 38
6.5.2 Consistency vs. Peak Performance . . . 39
6.5.3 Advantages of a Solver-Based Approach . . . 39
6.5.4 Limitations . . . 39
6.5.5 Threats to Validity . . . 40

7 Conclusion and Future Work **41**
7.1 Conclusion . . . 41
7.2 Recommendations for the Logistics Service Provider . . . 42
7.3 Future Research Directions . . . 42

References **45**

A Core Implementation Listings **46**

B Automated Infeasibility Diagnosis **49**
B.1 Theoretical Background . . . 49
B.2 Implementation Strategy . . . 49
B.2.1 Mapping Inputs to the Zeller Model . . . 49
B.2.2 Process Isolation . . . 50
B.2.3 Algorithmic Logic . . . 50
B.3 Output: The Toxic Scenario . . . 50
B.4 AI-Assisted Remediation . . . 51

C Detailed Experimental Results **52**
C.1 Phase 1 Results . . . 52
C.2 Phase 2 Results . . . 56
C.3 Extended Baseline SA Results . . . 56

Nomenclature

Abbreviations

Abbreviation	Definition
ARPD	Average Relative Percentage Deviation
CP-SAT	Constraint Programming – Satisfiability
GLS	Guided Local Search
LCI	Local Cheapest Insertion
LNS	Large Neighbourhood Search
MCS	Minimum Call Size
MILP	Mixed Integer Linear Programming
PCI	Parallel Cheapest Insertion
SA	Simulated Annealing
TEU	Twenty-foot Equivalent Unit
TS	Tabu Search
VRP	Vehicle Routing Problem
VRPPDTW	Vehicle Routing Problem with Pickup and Delivery and Time Windows

1
Introduction

1.1. Context and Motivation

The waterway network of Northwestern Europe plays a central role in global container logistics. Every year, millions of containers flow between major deep-sea ports such as Rotterdam and Antwerp and a dense hinterland of inland terminals across the Netherlands, Belgium, and Germany [1]. Barges, with average loading capacities exceeding 1,500 tonnes per vessel [2], offer a cost-effective and sustainable alternative to road freight.

Scheduling these vessels is a complex combinatorial problem. A planner must simultaneously consider heterogeneous vessel capacities, strict pickup-and-delivery precedence, time windows at both origin and destination, terminal opening hours, minimum call size requirements imposed by terminals, and multi-dimensional capacity constraints spanning volume, weight, refrigerated and dangerous goods. In practice, this task is performed manually by experienced logistics planners who rely on tacit knowledge accumulated over years of operations. While effective for routine situations, manual planning is time-consuming, difficult to scale, and inherently limited in its ability to globally optimize fleet utilization across dozens of vessels and hundreds of orders.

Automating this process promises substantial cost savings. Even modest reductions in total travel distance translate directly into lower fuel consumption, reduced emissions, and improved service reliability. However, the underlying optimization problem, classified as a Heterogeneous Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW), is NP-hard, meaning no efficient exact algorithm is known. This motivates the use of heuristic and metaheuristic approaches that can find high-quality solutions within the tight operational time budgets available to planners.

1.2. Company and Problem Description

This thesis was conducted in collaboration with a Dutch technology company specializing in logistics software for the intermodal transport sector. Their flagship product is an Enterprise Resource Planning (ERP) platform that supports terminal management, yard operations, gate automation, and vessel planning. Within this ecosystem, the vessel planning module assists operators in scheduling container movements across their fleet. The company serves multiple logistics operators, each with distinct fleet compositions, network topologies, and operational constraints.

At the time of this research, the company had developed an automated planning system based on a custom Simulated Annealing metaheuristic (referred to as Baseline SA throughout this thesis). This system constructs an initial solution using a randomized greedy insertion heuristic, then iteratively improves it through swap, insert, and rearrange neighbourhood operators governed by an adaptive temperature schedule. While Baseline SA is functional, it is not yet deployed in production; operators currently rely on manual planning, with Baseline SA serving as a candidate for automation. In our evaluation, Baseline SA exhibited two practical limitations:

1. **High variance across runs:** As an inherently stochastic method, Baseline SA can converge to

substantially different solutions depending on the random seed. In our experiments, the gap between the best and worst run reached up to 37.5% on the same input scenario, meaning operators cannot trust a single run to deliver a reliably good result.

2. **Difficulty of extension:** Adding new business constraints (e.g., water level restrictions, overtime call size rules) requires implementing penalty logic within the evaluation function and, in some cases, updating feasibility pre-filters, a process that is coupled to the internal solver architecture.

These limitations motivated the search for an alternative approach that could deliver consistent, high-quality solutions while providing a more modular framework for incorporating new constraints.

1.3. Research Goal and Objectives

The primary goal of this thesis is to develop and evaluate an improved scheduling system for the vessel scheduling problem using a modern constraint-based routing solver. Specifically, we aim to determine whether Google OR-Tools, an open-source optimization library combining Constraint Programming with Local Search metaheuristics, can outperform both human planners and the existing Baseline SA solver.

This goal is decomposed into the following objectives:

1. **Model:** Formulate the vessel scheduling problem as a Heterogeneous VRPPDTW with domain-specific side constraints (minimum call size, forbidden terminals, mandatory breaks, terminal opening hours) within the Constraint Programming paradigm.
2. **Implement:** Build a complete solver pipeline, named *Orion*, using the Google OR-Tools Routing Library, translating the mathematical formulation into an operational system capable of processing real-world planning scenarios.
3. **Evaluate:** Systematically evaluate the solver by filtering construction heuristics, evaluating metaheuristic configurations, and benchmarking the best configuration against baselines.
4. **Compare:** Conduct a head-to-head comparison of Orion against human planners and Baseline SA across six real-world scenarios from three distinct logistics operators, analyzing both solution quality and operational reliability.

1.4. Research Questions

The research objectives give rise to the following questions:

RQ1: How does the solver-based approach compare to the existing Baseline SA and human planners in terms of travel cost across diverse operational scenarios?

RQ2: What is the impact of different construction heuristics and local search metaheuristics on solver performance, and which configuration yields the best results?

RQ3: How does the solver-based approach compare to Baseline SA in terms of solution consistency and operational reliability?

1.5. Thesis Outline

The remainder of this thesis is structured as follows:

Chapter 2 –Problem Context introduces the domain of container shipping, describes the operational entities and constraints, classifies the problem as a Heterogeneous VRPPDTW, and discusses its computational complexity.

Chapter 3 –Theoretical Framework contrasts Mixed Integer Linear Programming with Constraint Programming, justifies the CP paradigm, and reviews the construction heuristics, local search operators, and metaheuristics that form the algorithmic foundation.

Chapter 4 –Problem Formulation provides the formal mathematical model, defining the sets, parameters, decision variables, objective function, and constraints.

Chapter 5 –Implementation details the technical realization of the Orion solver using Google OR-Tools, including the data preprocessing pipeline, constraint modelling strategies, and the architecture of the baseline solvers.

Chapter 6 –Evaluation presents the three-phase experimental evaluation: feasibility analysis, parameter tuning, and benchmarking against human planners and Baseline SA.

Chapter 7 –Conclusion and Future Work summarizes the findings, answers the research questions, and identifies directions for future research.

2
Problem Context

2.1. Waterborne Container Transport

Container shipping plays a pivotal role in the logistics network of Northwestern Europe. This thesis addresses two distinct but operationally related segments of this domain: inland shipping and short-sea shipping.

Inland waterway transport connects the major seaports of Rotterdam and Antwerp with the hinterland via a dense network of inland waterways. Barges operate at lower speeds compared to trucks, but their ability to transport large volumes, with average loading capacities exceeding 1,500 tonnes per vessel [2], makes them a highly efficient mode of transport for non-time-critical freight. The primary operational flow involves the transportation of containers between large sea terminals and smaller inland terminals, supporting the 'modal shift' from road to water to reduce congestion and improve supply chain sustainability.

Short-sea shipping extends this network across open water, connecting continental ports to offshore markets such as the United Kingdom. These operations employ larger, sea-going vessels with distinct scheduling requirements compared to pure inland barges.

In the context of this thesis, we use data provided by three distinct logistics operators. To maintain confidentiality, they are anonymized as follows:

- **Operator A:** A hinterland operator managing long-haul barge corridors connecting deep-sea ports to inland terminals across the Benelux region. Their fleet is composed of standard barges and coupled units.
- **Operator B:** A short-sea logistics provider coordinating high-volume container and trailer transport between the Netherlands and the United Kingdom via short-sea routes. Their operations involve larger, sea-going vessels with distinct scheduling requirements.
- **Operator C:** An intra-port shuttle operator specializing in high-frequency, short-distance container transfers within a major port cluster.

2.1.1. Current Planning Process

Currently, the planning process for these operators is performed entirely manually by a team of human planners. Each day, these experts continually review incoming transport orders, assigning them to the available vessels based on their tacit knowledge of the network and vessel capabilities. While effective for standard operations, this manual approach is time-consuming and relies heavily on the experience of individual planners, making it difficult to globally optimize the fleet's efficiency or quickly adapt to large-scale disruptions. This thesis aims to automate and enhance this decision-making process.

2.1.2. Operational Entities

The vessel scheduling problem is defined by the interaction between three distinct entities: terminals, vessels, and orders.

Vessels The fleet is heterogeneous, meaning the vessels vary significantly in their attributes and operational areas. The fleet is composed of the operator's own vessels and a flexible pool of Charter Vessels. These can be categorized into:

- **Inland Fleet:** Ranging from small vessels (e.g., 80m, 54 TEU) suitable for restricted waterways, up to large Coupled Units (e.g., 178m, 342 TEU).
- **Short-Sea Fleet:** Used by the short-sea operator, these vessels are significantly larger (e.g., 500+ TEU) and sea-worthy, allowing them to cross the channel.

These distinct classes imply different constraints; for instance, short-sea vessels cannot access small inland terminals due to draft and length restrictions, while inland barges cannot cross to the UK.

Orders A transport request, or *order*, represents a specific container that must be moved through the network. Each order is defined by a specific Origin (Pickup) location and a Destination (Delivery) location. Crucially, orders are bound by strict time windows for both pickup and delivery, requiring synchronization between the vessel's route and the terminal's schedule.

Terminals The network consists of locations distributed across the Netherlands and Belgium, with a few in France and Germany, featuring a high concentration of terminals in Antwerp and Rotterdam. These are categorized into Sea Terminals (e.g., Rotterdam, Antwerp) and Inland Terminals. Sea terminals are high-volume hubs where large sea vessels discharge containers for the hinterland, often subject to significant congestion. Inland terminals serve as regional hubs for collecting export containers. A critical operational factor is that terminals are generally not available 24/7; they enforce specific Opening Hours (e.g., 06:00 to 22:00), restricting the times at which a vessel can be serviced.

2.1.3. Operational Constraints

While container shipping offers significant capacity advantages, it is subject to complex operational constraints that distinguish it from other transport modes.

Minimum Call Size (MCS) To maximize efficiency and prevent congestion at busy maritime terminals, operators enforce a 'Minimum Call Size' (MCS). This constraint implies that a vessel should ideally perform a minimum number of container moves (loading or discharging) per visit to justify the berthing process. In practice, failing to meet this threshold does not necessarily incur a direct financial fine, but rather affects the vessel's priority; terminals may deny a berth during peak times or delay the vessel until the volume justifies the stop. In mathematical models, this constraint is often strictly enforced via penalty costs to simulate the high operational friction of servicing small volumes.

The Port of Antwerp provides a relevant example of this constraint in practice. To increase the efficiency of container barge transport, the port established initiatives to consolidate small volumes. After a trial period where the MCS was set at thirty moves, the threshold was adjusted to twenty moves. This adjustment was made partly due to persistent low water levels in the rivers, which limit the loading capacity of barges. The consolidation pilot project successfully reduced the number of calls at maritime terminals by approximately forty percent while increasing the average call size by twenty-five percent [3].

Minimum Overtime Call Size (MOCS) In addition to the standard MCS, some terminals operate with extended opening hours that are subject to stricter volume requirements. For example, a terminal might have regular opening hours from 09:00 to 15:00, during which the standard MCS applies. However, it may also offer 'Overtime' service windows, such as 06:00 to 09:00 and 15:00 to 20:00. To use these overtime slots, a vessel must guarantee a significantly higher volume of moves (e.g., 100 moves instead of 10). This 'Minimum Overtime Call Size' ensures that the extra labor costs incurred by the terminal during non-standard hours are justified by the volume handled.

Fixed Appointments To manage terminal access, operators use 'Fixed Appointments'. These are pre-booked time windows at specific terminals that obligate the vessel to visit within the agreed timeframe and perform up to a specified number of moves. These should not be confused with 'Fixed Stops' (described below); fixed appointments are scheduling commitments tied to cargo handling, whereas fixed stops are mandatory constraints arising from fleet management needs. In the formal model (Chapter 4), they are defined as mandatory visits with associated move requirements. However, fixed appointments are not implemented in the current solver; their integration is discussed as future work in Chapter 7.

Fixed Stops Distinct from transport orders, planners may sometimes strictly mandate a vessel's presence at a specific location at a specific time. These 'Fixed Stops' are mandatory constraints that arise from planned cargo handling needs: a vessel must visit a specific terminal within a given window to perform a pre-agreed set of loading or discharging moves. Regardless of the optimal route for cargo, the solver must ensure the vessel visits these fixed locations within their specified time windows.

Water Level Restrictions Inland waterways, particularly along the Rhine corridor, are subject to fluctuating water levels that directly constrain vessel operations. When water levels drop, the maximum permissible draft decreases, forcing barges to reduce their cargo load to navigate shallow segments safely. Since the relationship between water level and permissible tonnage depends on the vessel's weight-to-draft ratio, even moderate fluctuations can drastically alter the effective capacity of a vessel on a given day. Planners routinely monitor water levels at known bottleneck locations (e.g., Kaub, Maxau, Ruhrort) and adjust loading plans accordingly. Although water levels in the Netherlands and Belgium are relatively consistent, they remain a relevant operational factor for all inland operators. This constraint is not modelled in the current thesis but is identified as a direction for future work.

2.2. The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a fundamental class of combinatorial optimization problems involving the distribution of goods between depots and customers. As defined by Toth and Vigo [4], the VRP can be described as a graph theoretic problem on a graph $G = (V, A)$, where $V = \{0, \ldots, n\}$ is the vertex set and A is the arc set. In this representation, vertex 0 generally corresponds to the depot, where the vehicles start and end their routes, while the remaining vertices $i = 1, \ldots, n$ correspond to the customers. A non-negative cost c_{ij} is associated with each arc $(i, j) \in A$, representing the travel cost or time required to traverse the link between two vertices.

The objective of the VRP is to determine a set of K simple circuits, or routes, such that:

1. Each circuit starts and ends at the depot vertex.
2. Each customer vertex is visited by exactly one circuit.
3. All operational constraints (e.g., capacity) are satisfied.
4. The global transportation cost is minimized.

The VRP generalizes the well-known Travelling Salesman Problem (TSP), which arises when $K = 1$ and capacity constraints are relaxed; consequently, the VRP is known to be NP-hard in the strong sense [4].

2.3. Classification of the Problem

By synthesizing the standard variants of vehicle routing, we classify the specific vessel scheduling problem addressed in this thesis as a Heterogeneous Vehicle Routing Problem with Pickup and Delivery and Time Windows (Heterogeneous VRPPDTW). This complex variant involves designing routes for a non-homogeneous fleet to satisfy paired transportation requests between specific origins and destinations, subject to capacity, precedence, and time window constraints.

The specific characteristics of our problem instance are defined as follows:

Heterogeneous Fleet Unlike the basic Capacitated VRP (CVRP) which assumes identical vehicles, our fleet is heterogeneous [4]. As detailed in the operational context, the fleet is composed of the operator's

own vessels and a flexible pool of charter vessels. These vessels differ not only in their capacity vectors but also in their cost structures, defined by varying sailing costs ($Cost_v^{km}$) and fixed daily usage costs ($Cost_v^{day}$).

Multi-Dimensional Capacity While standard VRP formulations often treat capacity as a single scalar, container transport imposes multi-dimensional constraints. Vessels are constrained by a capacity vector $Cap_v = \langle C_v^{TEU}, C_v^{Weight}, C_v^{Reefer}, C_v^{DG} \rangle$, representing limits on volume (TEU), weight, refrigerated containers, and dangerous goods. Feasibility requires checking all four dimensions simultaneously. In practice, vessels may also have separate capacity limits for 45-foot containers, which occupy more deck space than standard 20- or 40-foot units; however, this dimension is left out of scope for this thesis.

Pickup and Delivery Precedence The problem is a direct instance of the VRP with Pickup and Delivery (VRPPD). Instead of simple deliveries from a depot, each order $o \in O$ implies a pickup task o^+ at an origin node and a delivery task o^- at a destination node. This introduces strict precedence constraints: the origin must be visited before the destination in the vessel's route sequence ($Index(o^+) < Index(o^-)$), and both must be served by the same vessel.

Time Windows Time strictness is critical in this formulation (VRPTW). Each order o possesses distinct time windows for both its origin and destination: $[E_o^+, L_o^+]$ for the pickup and $[E_o^-, L_o^-]$ for the delivery. Furthermore, the network contains terminals that are not available 24/7. These location-specific opening hours ($Open_{l,day}, Close_{l,day}$) effectively impose additional time constraints on any visit to a specific terminal node.

Domain-Specific Side Constraints Our specific application imposes side constraints rarely found in the canonical literature. Most notably, the *Minimum Call Size (MCS)* constraint requires that if a vessel visits a terminal, the total number of moves performed must meet a threshold MCS_l to justify the berthing process. Additionally, the 'Fixed Appointments' mechanism (mandatory pre-booked visits with specified move requirements) adds another layer of complexity to the scheduling decision.

2.4. Computational Complexity

The Vehicle Routing Problem and its variants are known to be *NP-hard* in the strong sense. Since the VRP generalizes the Travelling Salesman Problem (TSP), finding an optimal solution requires computational time that grows exponentially with the number of customers. The addition of Pickup and Delivery constraints, along with strict Time Windows (VRPPDTW), significantly increases the problem's intricacy compared to a standard CVRP.

While the constraints in a VRPPDTW (such as tight time windows) reduce the number of *feasible* solutions, they make the task of finding *any* feasible solution significantly more difficult. The presence of coupling constraints (pickup must precede delivery) and the heterogeneous nature of the fleet creates a combinatorial explosion in the search space.

Due to this complexity, exact solution methods (such as Branch-and-Cut or Column Generation) are generally limited to small-scale instances or specific academic benchmarks [4]. In the context of the Northwestern European shipping network, the problem involves high-volume container flows (hundreds of orders) and complex side constraints like Minimum Call Sizes. Solving an instance of this magnitude to mathematical optimality is computationally intractable within the short operational planning windows available to planners.

2.5. Conclusion

This chapter has defined the vessel scheduling problem as a Heterogeneous VRPPDTW with domain-specific constraints. Given the operational requirements for speed and the scale of the combinatorial search space, standard exact algorithms are ill-suited for this application. Consequently, the following chapter, *Theoretical Framework*, will explore the heuristic and metaheuristic approaches required to solve this problem efficiently in a commercial environment.

3
Theoretical Framework

3.1. Mathematical Modelling Frameworks

The choice of a mathematical modelling framework is fundamental to the design of any optimization solver, as it dictates how business rules are translated into algorithmic constraints. This section contrasts the traditional Mixed Integer Linear Programming (MILP) approach with the Constraint Programming (CP) paradigm selected for this thesis, justifying the latter based on the specific complexities of the vessel scheduling problem.

3.1.1. Mixed Integer Linear Programming

Mixed Integer Linear Programming (MILP) is a standard approach in Operations Research for modelling Vehicle Routing Problems (VRP) [4]. In a typical MILP formulation, the problem is described using a linear objective function subject to a set of linear inequalities. Decision variables are often binary ($x_{ij} \in \{0, 1\}$), indicating whether a vessel travels directly from node i to node j.

While effective for standard VRPs, MILP faces significant challenges when modelling the complex conditional logic required for vessel scheduling.

- **Reliance on Linear Inequalities:** MILP solvers (e.g., CPLEX, Gurobi) rely on linear relaxation to bound the search space.
- **The 'Big-M' Weakness:** To model logical implications, such as 'If a vessel visits terminal l, it must handle at least X containers', MILP requires the use of 'Big-M' constraints (e.g., $y \leq M \cdot x$). As noted in the problem formulation, this approach often results in weak linear relaxations. Large values of M expand the solution space artificially, preventing the solver from effectively pruning the branch-and-bound tree and causing slow convergence for large instances ($N > 100$).
- **Combinatorial Explosion:** Exact methods like MILP suffer from a factorial growth in complexity. Modelling a VRP requires binary variables for every potential arc (x_{ij}) and time step. As the number of nodes N increases, the state space expands as $O(N!)$, making it computationally intractable for operational instances with hundreds of orders and complex side constraints.

3.1.2. Constraint Programming

In contrast to MILP, Constraint Programming (CP) is a paradigm focused on feasibility and logical relations rather than pure numerical optimization. As described by Shaw [5], CP is particularly well-suited for vehicle routing problems characterized by complex side constraints, such as legislative driver breaks or complex pay provisions.

Logical Predicates and Set Theory

Instead of forcing logic into linear equations, CP allows the problem to be defined using natural logical predicates and set theory. For example, the 'Forbidden Terminal' constraint can be modeled directly

by removing specific values from the domain of a variable, rather than adding binary variables to the matrix.

Domain Reduction (Propagation)

The primary algorithmic engine of CP is Domain Reduction (or Propagation). Unlike MILP, which finds an optimal solution within a relaxed polytope, CP actively removes values that cannot be part of a feasible solution.

- **Mechanism:** If a vessel v has a capacity of 100 TEU, and the solver assigns a route segment with a load of 120 TEU, the propagation engine immediately detects the conflict. It prunes the entire branch of the search tree associated with that assignment without needing to solve a linear sub-problem.
- **Application to Vessel Scheduling:** This mechanism is critical for handling the strict time windows and capacity dimensions of our problem. For instance, if a vessel is assigned a 'Forbidden Terminal,' the solver reduces the domain of the vehicle variable for that node, effectively pruning the search space at the root.

3.1.3. Handling Constraints: Hard vs. Soft

A critical aspect of the modelling framework is the distinction between hard and soft constraints, which directly impacts the solver's ability to find solutions in a highly constrained environment.

Hard Constraints

Hard constraints are non-negotiable rules that determine the feasibility of a solution. If a hard constraint is violated, the solution is discarded as invalid. In our vessel scheduling model, physical limitations such as vessel capacity and strict precedence (pickup must occur before delivery) are modeled as hard constraints.

Soft Constraints and Penalties

Soft constraints are rules that should be respected but can be violated at a cost. Instead of discarding the solution, the solver adds a penalty term to the objective function.

This distinction is vital for the operational robustness of the solver. As detailed in Chapter 5, the Minimum Call Size (MCS), a rule requiring a minimum number of moves per terminal visit, was implemented as a soft constraint with a penalty gradient.[1]

- **Rationale:** Modelling MCS as a hard constraint often leads to 'No Solution' results in sparse schedules.
- **Penalty Gradient:** By converting it to a soft constraint with a linear penalty gradient, we ensure the solver can always return a valid schedule, even if it is operationally imperfect. This 'smooths' the search space, providing the local search heuristic with a gradient to follow toward optimization rather than confronting it with a binary wall of infeasibility.

3.2. Solution Methodologies

Having defined the vessel scheduling problem as a Heterogeneous VRPPDTW with complex side constraints in Chapter 2, we now turn to the algorithmic engines required to solve it. The selection of a solution methodology is driven by a trade-off between solution quality (optimality) and computational feasibility (scalability).

As illustrated in Figure 3.1, solution methodologies for Vehicle Routing Problems are generally classified into three categories: Exact Methods, Constructive Heuristics, and Metaheuristics [6]. While exact methods provide optimality guarantees, the NP-hard nature of the VRPPDTW necessitates the use of heuristic approaches for large-scale operational instances.

It is worth noting that a fourth category, *Data-Driven approaches* (e.g., Deep Reinforcement Learning), has recently emerged. While promising, these methods typically require massive historical datasets for

[1] The MCS penalty was ultimately disabled in the final evaluation; see Chapter 5 for details.

training and act as 'black boxes' that are difficult to debug in an operational setting where explainability is key. Therefore, they are considered out of scope for this thesis.

This section outlines the theoretical hierarchy of methods considered, identifying the construction heuristics and metaheuristics available in the OR-Tools framework for empirical evaluation in Chapter 6. The final configuration is determined empirically in Chapter 6.

3.2.1. Exact Methods

Exact solution methods, such as Branch-and-Bound, Branch-and-Cut, or Branch-and-Price, guarantee finding the globally optimal solution. In the domain of Vehicle Routing, the most successful exact approach is currently Branch-and-Price, which combines Branch-and-Bound with Column Generation [7].

In this framework, the master problem (usually a Set Partitioning formulation) selects the best combination of routes, while the pricing sub-problem (usually a Shortest Path Problem with Resource Constraints) generates valid routes to improve the master solution [8]. While effective for standard problems, this approach becomes computationally intractable for the specific problem defined in this thesis due to three compounding factors:

- **Scale:** Exact methods face severe scalability limits. As noted by Uchoa et al. [9], while recent Branch-Cut-and-Price algorithms have managed to solve standard CVRP instances up to 275 customers, this success relies on specific conditions (e.g., short routes). For general cases, and particularly for 'rich' variants like the VRPPDTW, the traditional barrier of 100 nodes remains a critical bottleneck. Since the operational instances in this thesis (Chapter 5) exceed 400 nodes, exact methods are computationally intractable.
- **Heterogeneous Fleet:** As defined in Section 2.3, our fleet is non-homogeneous. In a Branch-and-Price context, this requires solving a distinct pricing sub-problem for every unique vehicle type, significantly increasing the computational overhead per iteration.
- **Multi-Dimensional Capacity:** The pricing sub-problem solves a shortest path problem on a graph where resource consumption (capacity) is tracked via labels. Our problem enforces capacity on four simultaneous dimensions (TEU, Weight, Reefer, Dangerous Goods). This high dimensionality prevents the dominance rules used in labeling algorithms from pruning the state space effectively.

Consequently, exact methods are dismissed. This decision aligns with the broader research trend; Braekers et al. [7] reveal that over 71% of recent VRP publications employ metaheuristics, compared to only 17% using exact approaches. Given the 'substantial complexity' of VRPPDTW variants [7] and the scalability limits evidenced by Uchoa et al. [9], heuristic approaches are the only viable path for the operational instances considered in this thesis.

3.2.2. Construction Heuristics

Heuristic solvers typically operate in two phases: constructing an initial feasible solution and then iteratively improving it. Several classic constructive heuristics exist in the literature, including the Nearest Neighbour method and the Sweep Algorithm [6].

Rejection of Sweep and Sequential Methods

The Sweep Algorithm [6] clusters customers based on their polar angle from the depot. While effective for Capacitated VRPs, it is ill-suited for the VRPPDTW. The precedence constraint (Pickup → Delivery) often conflicts with angular clustering; a pickup node may be angularly 'after' its delivery node, breaking the sweep logic.

Similarly, Sequential Insertion (Nearest Neighbour) is dismissed. This greedy approach builds one route at a time, filling a vehicle to capacity before initializing the next. As observed by Potvin and Rousseau [10], this approach often leaves the 'hardest' customers (those with tight time windows or remote locations) for the last route. In a VRPPDTW, this often leads to infeasibility because the remaining unrouted orders cannot fit into the remaining time or capacity slots.

Cheapest Insertion Heuristics

To mitigate the 'garbage collection' issues of sequential methods, cheapest insertion heuristics evaluate the cost impact of each insertion rather than greedily extending a single route. Two variants are available in OR-Tools:

- **Parallel Cheapest Insertion (PCI):** Proposed by Potvin and Rousseau [10], PCI initializes multiple routes simultaneously and, in each iteration, evaluates the insertion cost of *every* unrouted order into *every* active route.
- **Local Cheapest Insertion (LCI):** A variant that inserts each order into the single route where it causes the least cost increase, processing orders one at a time.

In both variants, the insertion cost is weighted by detour distance, delay time, and waiting time:

$$Cost(u) = \alpha_1 \times \text{Detour}(u) + \alpha_2 \times \text{Delay}(u) + \alpha_3 \times \text{Wait}(u) \tag{3.1}$$

The global evaluation capability of these heuristics is critical for our application. By considering multiple routes, they avoid painting the solver into a corner, ensuring that orders with tight precedence constraints or narrow time windows (Chapter 4) are integrated into the schedule early, rather than being forced into a final, infeasible route. The relative performance of PCI and LCI is compared empirically in Chapter 6.

Algorithm 1 Parallel Cheapest Insertion (PCI) with Generalized Regret

Require: Set of customers U, Number of routes N
1: Initialize N routes with N seed customers (farthest from depot)
2: **while** $U \neq \emptyset$ **do**
3: **for** each unrouted customer $u \in U$ **do**
4: **for** each route $r \in \{1, \dots, N\}$ **do**
5: Calculate best insertion cost $c^*_{1r}(u)$
6: **end for**
7: Calculate Generalized Regret: $c_2(u) = \sum_{r \neq r^*}(c^*_{1r}(u) - c^*_{1r^*}(u))$
8: **end for**
9: Select customer u^* with $\max(c_2(u))$
10: Insert u^* into route r^* at best position
11: **if** Insertion is infeasible **then**
12: **break** (Declare instance infeasible or add route)
13: **end if**
14: $U \leftarrow U \setminus \{u^*\}$
15: Update route constraints (time windows, capacity)
16: **end while**

3.2.3. Improvement Heuristics: Local Search

Construction heuristics rarely yield near-optimal solutions. To reduce the objective function, we apply Improvement Heuristics, commonly known as Local Search. This process iteratively explores the 'neighbourhood' of the current solution by applying small modifications.

Standard Neighbourhood Operators

The implementation uses the Google OR-Tools routing library [11], which employs a portfolio of neighbourhood operators to iteratively improve the solution. Rather than relying on a single move type, the solver dynamically selects from a set of standard operators.

The core neighbourhood structures can be categorized into three atomic movements [6]:

- **Relocate:** Moves a node from one route to another.
- **Swap:** Exchanges the positions of two nodes.
- **2-Opt:** Removes two edges and reconnects the sequence (effective for untangling crossing paths).

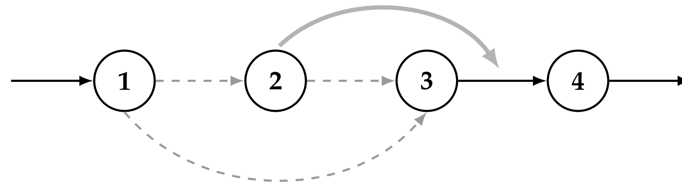


Figure 3.2: Illustration of the Relocate (intra-route) operator. Node 2 is removed from its sequence and reinserted between Node 3 and Node 4.

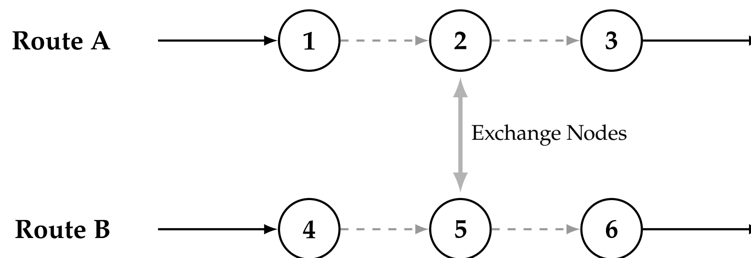


Figure 3.3: Illustration of the Swap (inter-route) operator. Node 2 from Route A is exchanged with Node 5 from Route B. The dashed edges are removed and replaced by new edges connecting the swapped nodes into their new routes.

Figure 3.4: Illustration of the 2-Opt (intra-route) operator. (a) The initial route follows the sequence $1 \to 2 \to 3 \to 4$, resulting in crossing edges. (b) The operator reconnects the nodes to form the non-crossing sequence $1 \to 3 \to 2 \to 4$. Note that the segment between nodes 2 and 3 has reversed direction.

While the literature proposes complex operators such as CROSS or GENI [6], modern routing engines like OR-Tools prioritize the use of these atomic $O(N^2)$ operators.

The rationale is that structural diversification is better handled by the Large Neighbourhood Search (LNS) framework rather than complex local moves. By delegating the 'global' shuffling of orders to LNS (detailed in the following section), the Local Search phase remains computationally efficient, focusing purely on fine-grained trajectory smoothing.

Large Neighbourhood Search

Standard local search operators often get trapped in local optima, particularly in highly constrained environments like the vessel scheduling problem. For example, moving a single container might be impossible due to the Minimum Call Size (MCS) constraint at a terminal, but moving a *batch* of containers might be valid.

To address this, we employ the Large Neighbourhood Search (LNS) principle described by Shaw [5]. LNS operates on a 'Ruin and Recreate' basis:

1. **Ruin:** A significant portion of the solution (e.g., 30% of orders) is removed based on heuristics (e.g., spatial proximity or randomness).
2. **Recreate:** The removed orders are re-inserted into the schedule using a constraint-based tree search.

As argued by Shaw [5], this allows the solver to traverse 'barriers' in the search space that would block standard operators. It effectively shuffles the solution to find better global structures without violating the strict capacity and precedence rules defined in Chapter 4.

3.2.4. Metaheuristics: Escaping Local Optima

Even with LNS, the search can stagnate. Metaheuristics are high-level strategies that guide the local search to escape local optima. As noted in Figure 3.1, these are broadly classified into Population-based methods (e.g., Genetic Algorithms, Ant Colony Optimization) and Single-solution methods (e.g., Simulated Annealing, Tabu Search, GLS) [6].

Rejection of Genetic Algorithms (Population-based)

Genetic Algorithms (GA) are popular in the literature but are rejected for this implementation. In a VRPPDTW, standard crossover operators (combining two parent routes) frequently generate infeasible offspring by breaking the Pickup → Delivery precedence chain or violating strict time windows [6]. Repairing these infeasible solutions requires expensive algorithms that drastically reduce the solver's iteration speed in an operational context.

Note on Hybrid Genetic Search: It must be acknowledged that recent advancements, such as the Hybrid Genetic Search (HGS) by Vidal et al. [12], have successfully combined GA with local search to achieve state-of-the-art results. However, implementing such a complex hybrid framework from scratch was deemed less efficient than leveraging the battle-tested, generalized heuristics provided by the OR-Tools library.

Candidate Single-Solution Metaheuristics

OR-Tools provides five single-solution metaheuristics: Greedy Descent (GD), Guided Local Search (GLS), Tabu Search (TS), Generic Tabu Search (GTS), and Simulated Annealing (SA). All five are evaluated empirically in Chapter 6; the three most theoretically distinct strategies are described below.

Tabu Search (TS). Tabu Search relies on short-term memory structures (a *tabu list*) to forbid reversing recent moves, preventing the search from cycling back to previously visited solutions [6]. By systematically exploring the neighbourhood while avoiding revisits, TS can traverse barriers in the search space that would trap a simple descent method. The primary tuning parameter is the tabu tenure (the number of iterations a move remains forbidden).

Simulated Annealing (SA). Simulated Annealing accepts worsening solutions with a probability governed by a temperature parameter that decreases over time [6]. At high temperatures, the search is exploratory; as the temperature cools, it becomes increasingly greedy. SA requires calibration of the initial temperature and cooling schedule, which can be sensitive to the problem instance.

Guided Local Search (GLS). Guided Local Search, proposed by Voudouris and Tsang [13], takes a different approach: rather than modifying the search trajectory, it modifies the objective function itself.

GLS augments the actual cost function $g(s)$ with penalty terms based on solution features (e.g., long edges). The augmented cost function $h(s)$ is defined as:

$$h(s) = g(s) + \lambda \sum_{i=1}^{M} p_i \cdot I_i(s) \tag{3.2}$$

where p_i is the current penalty counter for feature i, and $I_i(s)$ is an indicator function (1 if feature i is present in solution s).

When the solver gets trapped in a local minimum, GLS identifies the features that contribute most to the cost (e.g., the longest travel leg) and increments their penalty p_i. This effectively 'fills in' the local valley, forcing the local search to move to a different part of the solution space. GLS relies principally on a single parameter λ (the penalty weight) and, as demonstrated by Voudouris and Tsang [13], adapts automatically to the landscape of the problem instance.

Algorithm 2 Guided Local Search (GLS) [13]

Require: Search space S, Objective function $g(s)$, Penalty weight λ, Features M
Ensure: Best solution found s^*
1: $k \leftarrow 0$
2: $s_0 \leftarrow$ GenerateInitialSolution(S)
3: Initialize penalties $p_i \leftarrow 0 \quad \forall i \in \{1, \ldots, M\}$
4: **while** StoppingCriterion is **False do**
5: Define augmented cost: $h(s) = g(s) + \lambda \sum_{i=1}^{M} p_i \cdot I_i(s)$
6: $s_{k+1} \leftarrow$ LocalSearch(s_k, h)
7: **for** $i \leftarrow 1$ **to** M **do**
8: $util_i \leftarrow I_i(s_{k+1}) \cdot \frac{c_i}{1+p_i}$
9: **end for**
10: **for each** feature i with maximal $util_i$ **do**
11: $p_i \leftarrow p_i + 1$
12: **end for**
13: $k \leftarrow k + 1$
14: **end while**
15: **return** s^* with lowest $g(s^*)$

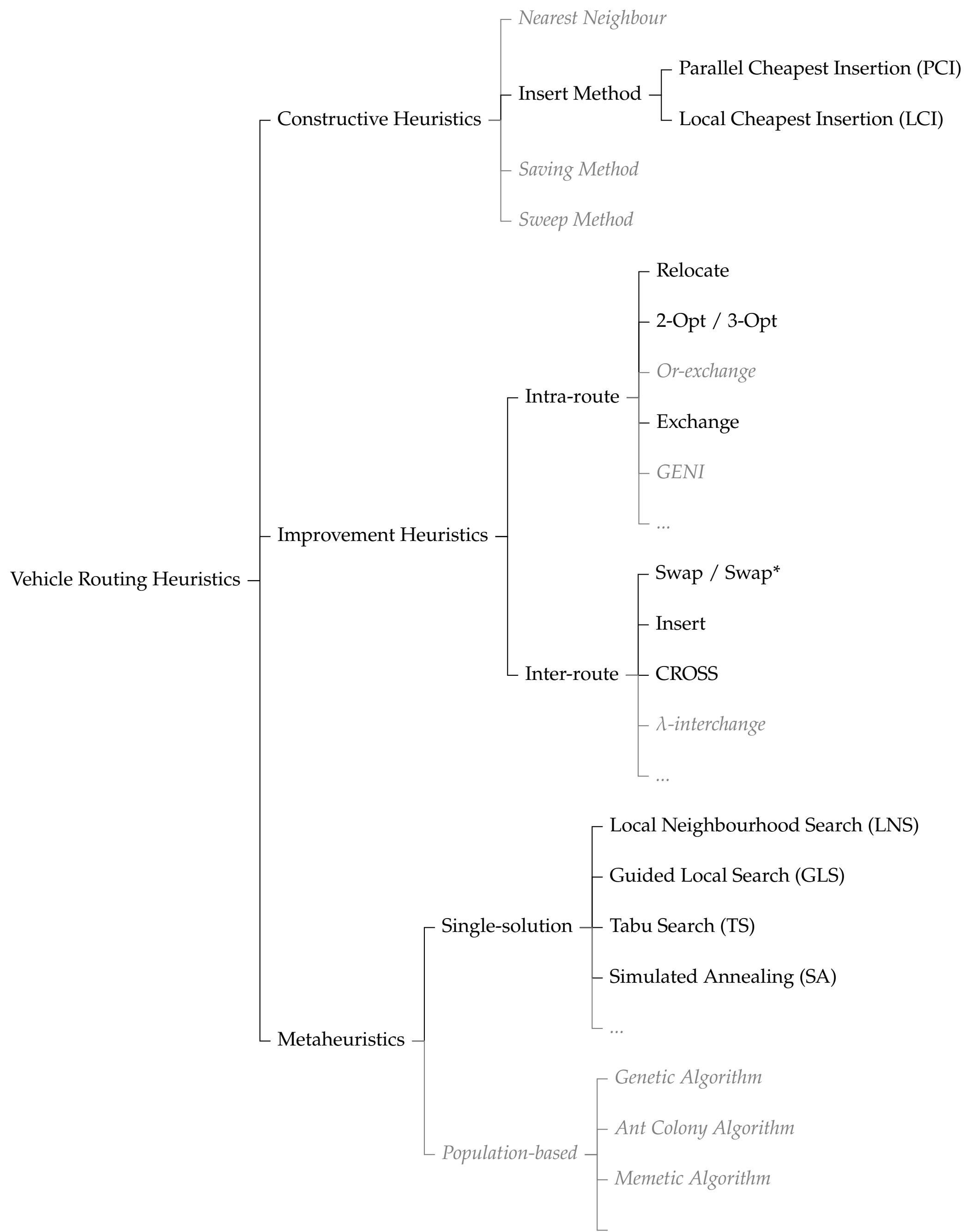


Figure 3.1: Taxonomy of VRP heuristics adapted from Liu et al. Standard black text indicates the candidate methods considered for evaluation; gray italic text represents alternative methodologies dismissed a priori.

4 Problem Formulation

4.1. Detailed Problem Definition

The core problem addressed in this thesis is the optimization of container vessel scheduling within an intermodal network. In the container shipping industry, specifically within the Benelux region, containers must be transported between large sea terminals (e.g., Rotterdam, Antwerp) and smaller inland terminals via waterways. This is a variation of the Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW), where the primary objective is to construct feasible schedules for a fleet of vessels to deliver containers from their origin to their destination within specified deadlines while minimizing operational costs.

The problem involves three distinct entities:

- **Orders:** These are transportation requests defined by a container that needs to be moved. Each order has a specific load location (origin) and discharge location (destination), along with strict time windows for both pickup and delivery. Furthermore, orders have physical characteristics that constrain transport, specifically TEU (Twenty-foot Equivalent Unit) size, weight, and boolean indicators for Reefer (refrigerated) and Dangerous Goods status. In practice, vessels may also distinguish 45-foot containers, which occupy more deck space than standard units; however, this dimension is left out of scope for this thesis.
- **Vessels:** The fleet consists of vessels with varying attributes. Each vessel has defined capacities for TEU, weight, reefers, dangerous goods, and 45-foot containers (the latter is excluded from the model in this thesis). Vessels also have specific cost structures (cost per kilometre and cost per day), availability windows, and restrictions on which terminals they can visit (forbidden terminals).
- **Terminals:** Locations in the network where containers are loaded or discharged. Terminals impose constraints such as opening times and handling costs. A critical operational constraint is the 'Minimum Call Size' (MCS). Terminals often require a minimum volume of container moves (loading or discharging) per vessel visit to justify the berthing process.

In practice, the problem also involves 'Fixed Appointments': mandatory pre-booked time windows at specific terminals that obligate the vessel to visit within the agreed timeframe and perform a specified number of moves. While Fixed Appointments are part of the operational reality described in Chapter 2, the formal model in this thesis handles only their simplified counterpart, 'Fixed Stops' (pre-assigned terminal visits without move requirements), and treats full Fixed Appointment logic as future work.

4.1.1. Illustrative Data Example

To clarify the input data structures, Table 4.1 presents a simplified 'toy' dataset of transport orders. Each order specifies a physical container with mandatory precedence: the origin (pickup) must be visited before the destination (delivery). The time windows dictate the valid service intervals at each location. For example, Order 1 requires moving a 20ft container from a terminal in Rotterdam to one in Antwerp, with the pickup scheduled for the morning of Day 1.

Table 4.1: Illustrative 'Toy Data' for Vessel Transport Orders

Order ID	Type	Location	Window Start	Window End	TEU
1	Pickup Delivery	Sea Terminal A Sea Terminal B	Day 1 08:00 Day 2 08:00	Day 1 12:00 Day 2 18:00	1.0
2	Pickup Delivery	Sea Terminal B Inland Terminal C	Day 1 14:00 Day 3 06:00	Day 1 20:00 Day 3 22:00	2.0
3	Pickup Delivery	Sea Terminal A Inland Terminal D	Day 1 09:00 Day 2 12:00	Day 1 15:00 Day 3 12:00	1.0

4.2. Formal Problem Description

We formulate the vessel scheduling problem using a Constraint Programming (CP) approach. Unlike Mixed Integer Linear Programming (MILP), which relies on linear inequalities and 'Big-M' notation to enforce logical conditions, this formulation uses set theory and logical predicates to describe the problem structure naturally.

4.2.1. Sets and Parameters

We define the following sets to represent the problem entities:

- $\mathcal{V}$: Set of available vessels.
- $\mathcal{O}$: Set of orders (transportation requests). Each order $o \in \mathcal{O}$ implies a pickup task o^+ and a delivery task o^-. A special subset $\mathcal{O}^{OB} \subset \mathcal{O}$ represents on-board orders (cargo already loaded on a vessel), which have only a delivery task o^-.
- $\mathcal{N}$: Set of nodes representing tasks, where $\mathcal{N} = \{o^+ \mid o \in \mathcal{O}\} \cup \{o^- \mid o \in \mathcal{O}\}$.
- $\mathcal{L}$: Set of physical locations (terminals).
- $\mathcal{T}$: The planning horizon, represented as a continuous time interval $[T_{start}, T_{end}]$.

Based on the business rules provided in the data processing definitions, we define the following parameters:

Vessel Parameters For each vessel $v \in \mathcal{V}$:

- Cap_v: A vector representing the vessel's capacity dimensions $\langle C_v^{TEU}, C_v^{Weight}, C_v^{Reefer}, C_v^{DG} \rangle$.
- $Depot_v$: The starting location of the vessel.
- $Forbidden_v \subseteq \mathcal{L}$: A set of terminals the vessel cannot visit. In practice, this is used by planners to enforce fleet segmentation (e.g., restricting specific vessels to specific regions), simplifying the planning process.
- $Cost_v^{km}$: The sailing cost per unit distance. Although a daily usage cost $Cost_v^{day}$ is defined in the data model, it is excluded from the active optimization due to inconsistent data on vessel ownership (see Note on Experimental Scope below).

Order Parameters For each order $o \in \mathcal{O}$:

- Req_o: A vector of requirements $\langle R_o^{TEU}, R_o^{Weight}, R_o^{Reefer}, R_o^{DG} \rangle$.
- Loc_o^+, Loc_o^-: The physical locations (terminals) for pickup and delivery.
- $[E_o^+, L_o^+]$: The time window (Earliest, Latest) for pickup.
- $[E_o^-, L_o^-]$: The time window (Earliest, Latest) for delivery.

Location Parameters For each location $l \in \mathcal{L}$:

- MCS_l: The Minimum Call Size (minimum moves required) for a visit.
- $Open_{l,day}, Close_{l,day}$: The opening and closing times for a specific day.
- $Cost_l^{call}$: The fixed cost for calling at terminal l. In practice, the primary real cost is a harbour entry fee incurred when a vessel first enters a port cluster, rather than a per-terminal charge. This parameter captures that cost and also serves to discourage unnecessary stops in the solver.
- $Dist(i,j), Time(i,j)$: The distance and travel time between any two locations $i, j \in \mathcal{L}$.

4.2.2. Decision Variables

In Constraint Programming, we use sequence variables rather than binary flow matrices.

- σ_v: A sequence variable representing the ordered list of nodes (stops) visited by vessel v. $\sigma_v = (n_1, n_2, \ldots, n_k)$ where $n_i \in \mathcal{N}$.
- τ_i: A continuous variable representing the arrival time at node $i \in \mathcal{N}$.
- λ_i: A vector variable representing the cumulative load on board the vessel after visiting node i.
- u_o: A boolean variable indicating if order o is unplanned (1 if unplanned, 0 if planned).

4.2.3. Objective Function

The objective is to minimize the total operational cost, which is a sum of transportation, usage, handling, and penalty costs.

$$\text{Minimize } Z = \sum_{v \in \mathcal{V}} (C_v^{Trans} + C_v^{Usage} + C_v^{Call}) + C^{Penalty} \tag{4.1}$$

where the components are defined as follows based on the specified cost structures:

- **Transportation Cost:** Based on the total distance sailed by the sequence σ_v.

$$C_v^{Trans} = Cost_v^{km} \times \sum_{i=1}^{|\sigma_v|-1} Dist(Loc_{\sigma_v[i]}, Loc_{\sigma_v[i+1]})$$

- **Usage Cost:** Incurred for every day the vessel is active.

$$C_v^{Usage} = Cost_v^{day} \times \text{DaysActive}(\sigma_v)$$

- **Call Cost:** The harbour entry fee incurred when a vessel enters a port cluster.

$$C_v^{Call} = \sum_{l \in Unique(\sigma_v)} Cost_l^{call}$$

- **Penalty Cost:** A high cost applied for every order left unplanned.

$$C^{Penalty} = \sum_{o \in O} u_o \times Penalty_{unplanned}$$

1

[1]While the formulation above presents the complete economic model including vessel usage (charter) costs and terminal handling costs, the specific evaluation in this thesis excludes C_v^{Usage} from the active optimization. Inconsistent data regarding vessel ownership (owned vs. rented) necessitates removing this variable to ensure a fair baseline comparison across all scenarios. Handling costs, being constant for a defined set of mandatory orders, are similarly treated as a constant offset rather than a decision variable.

Refinement of Cost Definitions It is important to qualify the economic realism of these terms:

- **Charter/Usage Costs:** In reality, a company owns many of its vessels. For these, the 'daily cost' is effectively a sunk cost (crew salaries are paid regardless of activity). True variable savings only occur when reducing the usage of *rented* (charter) vessels. As noted below, this term is excluded from the active optimization.
- **Call Costs:** The primary real cost captured here is the harbour entry fee incurred when a vessel first enters a port cluster. Individual terminals within the same port do not charge separate visit fees. In our experiments, we approximate C_v^{Call} with a uniform fixed penalty of 100 per stop, which serves both to represent the harbour fee and to discourage unnecessary stops.
- **Handling Costs:** Although technically a real cost per container, in our dataset these are treated as zero or fixed constants, as they do not vary based on the routing decision (the container must be moved regardless of the route).

4.2.4. Operational Constraints

We formulate the constraints using logical predicates and set membership to ensure feasibility without relying on linear relaxations.

Routing and Precedence

Every planned order must have its pickup and delivery nodes assigned to the same vessel's sequence, and the pickup must precede the delivery.

1. **Vessel Assignment Logic:** If an order o is planned ($u_o = 0$), there must exist exactly one vessel v that visits both the pickup node o^+ and delivery node o^-.

$$u_o = 0 \implies \exists v \in \mathcal{V} : \{o^+, o^-\} \subset \sigma_v$$

2. **Precedence constraint:** For any planned order o served by vessel v, the index (position) of the pickup node in the sequence must be strictly less than the index of the delivery node.

$$\forall v \in \mathcal{V}, \forall o \in \mathcal{O} : \{o^+, o^-\} \subset \sigma_v \implies Index(o^+, \sigma_v) < Index(o^-, \sigma_v)$$

3. **Forbidden Terminals:** A vessel cannot visit a node if its location is in the vessel's forbidden set.

$$\forall v \in \mathcal{V}, \forall n \in \sigma_v : Loc_n \notin Forbidden_v$$

Time Constraints

Transitions between nodes must respect travel times and service durations, and visits must occur within time windows.

1. **Transition Logic:** For any two consecutive nodes i and $i+1$ in sequence σ_v, the arrival time at $i+1$ must be greater than or equal to the departure from i plus travel time.

$$\tau_{i+1} \geq \tau_i + ServiceTime_i + Time(Loc_i, Loc_{i+1})$$

2. **Time Windows:** The arrival time at any node n (pickup or delivery) must fall within the specific time window of the associated order o.

$$\forall o \in \mathcal{O} : \tau_{o^+} \in [E_o^+, L_o^+] \land \tau_{o^-} \in [E_o^-, L_o^-]$$

   Additionally, the arrival must respect the opening hours of the terminal location Loc_n.

Capacity Constraints

The load on board must never exceed the vessel's capacity across all four dimensions (TEU, Weight, Reefer, Dangerous Goods).

Let Δ_n be the change in load at node n (positive for pickup Req_o, negative for delivery $-Req_o$). The cumulative load λ at step k of sequence σ_v is defined recursively:

$$\lambda_{\sigma_v[k]} = \sum_{j=1}^{k} \Delta_{\sigma_v[j]}$$

The constraint is strictly defined as:

$$\forall v \in \mathcal{V}, \forall k \in \{1 \ldots |\sigma_v|\} : \lambda_{\sigma_v[k]} \leq Cap_v$$

Minimum Call Size

The Minimum Call Size constraint is conditional. If a vessel visits a terminal, the total number of moves (containers loaded + containers discharged) at that terminal during that visit must meet the threshold MCS_l.

Let $Moves(v, l)$ be the set of order nodes in sequence σ_v that are located at terminal l.

$$Moves(v, l) = \{n \in \sigma_v \mid Loc_n = l\}$$

The constraint is formulated as a logical implication:

$$\forall v \in \mathcal{V}, \forall l \in \mathcal{L} : Moves(v, l) \neq \emptyset \implies |Moves(v, l)| \geq MCS_l$$

This ensures that the vessel only docks at terminal l if it performs enough work to satisfy the terminal's requirements.[2]

4.3. Model Assumptions

The formulation above relies on several simplifying assumptions. These are deliberate modelling choices made to keep the problem tractable; their implications for result interpretation are revisited in Section 6.5.4.

4.3.1. Wild Anchorage

The model assumes that a vessel can stop at any point along a waterway when its daily operating time expires (e.g., at 22:00) and resume immediately from that location the next morning (e.g., at 06:00). In reality, inland vessels are legally required to dock at designated overnight parking locations, which may be kilometres away from their stopping point. This assumption is relevant only for hinterland operators whose vessels observe daily operating windows. For short-sea operators (e.g., Operator B in Chapter 6), vessels operate continuously around the clock, so this assumption does not apply.

We deliberately exclude specific parking logic to avoid the combinatorial explosion associated with modelling explicit 'Wait Nodes' or Time-Expanded Graphs (TEG). To counterbalance this optimistic assumption (which effectively creates "teleportation" to a parking spot), the model uses conservative fixed speeds that underestimate actual vessel capabilities. This buffer ensures that the schedules produced, despite the wild anchorage assumption, remain achievable in practice, as confirmed by feedback from human planners.

4.3.2. Deterministic Travel Times

The model assumes a fixed speed per vessel and does not account for traffic, weather, or lock waiting times. A single speed value is used uniformly for all vessels and routes. In practice, vessels may travel faster or slower depending on waterway conditions, load, and operational priorities.

[2] While the formulation above presents MCS as a hard logical implication for mathematical clarity, Chapter 5 describes a soft-constraint implementation with a linear penalty gradient. As detailed in Section 5.5, this constraint was ultimately disabled for the final evaluation, as the solver naturally achieved efficient terminal clustering through travel distance minimization alone.

4.3.3. Static Capacity

The formulation treats vessel capacity as a fixed parameter for the entire planning horizon. In reality, fluctuating water levels along inland rivers impose dynamic capacity constraints on barges. As described in Chapter 2, historical data from ELWIS indicates that the Kaub water level alone varied between 73 cm and 337 cm over a single year (2025–2026), with an average weekly fluctuation of over 59 cm. Depending on the vessel's weight-to-draft ratio, these fluctuations drastically alter the maximum container tonnage a vessel can carry through the bottleneck. Since this constraint varies dynamically over time and geography, it cannot be adequately captured by a single static capacity limit. While it is technically possible to preprocess a worst-case static bound (e.g., reducing vessel capacity to the minimum projected water level across the planning horizon), this would be overly conservative and result in heavily underutilised schedules.

4.3.4. Fixed Drop Penalty

Unassigned orders are penalized with a fixed, high cost to force the solver to discover feasible routes. In a true intermodal optimization context, the drop cost should dynamically reflect the real-world alternative: the cost to transport that specific container via trucking or rail. The current formulation does not model this distinction.

5

Implementation with Google OR-Tools

This chapter details the technical implementation of the VRPPDTW solver using the Google OR-Tools framework [11]. We discuss the architectural pivot from CP-SAT to the Routing Library, the data preprocessing pipeline, and the specific constraint modelling strategies used to handle the complex business rules defined in Chapter 4.

5.1. Architecture and Design Rationale

The selection of the solver backend was a critical architectural decision driven by a trade-off between expressiveness and scalability.

5.1.1. Strategic Choice: Why Google OR-Tools?

The decision to use the Google OR-Tools Routing Library, rather than developing a custom metaheuristic from scratch or using a commercial black-box solver, is grounded in four strategic advantages:

- **Reproducibility and Standardization:** Unlike custom implementations which are often 'black boxes' dependent on the specific coding skills of the researcher, OR-Tools is a widely-used, open-source standard. This ensures that the results presented in this thesis are reproducible by other researchers and companies.
- **Accessibility and Cost:** As an open-source library (Apache 2.0 license), it removes the high financial barrier typical of commercial solvers (e.g., CPLEX, Gurobi). This is particularly relevant for the target audience of this research, as it removes a potential barrier to adoption for shipping operators evaluating automated planning tools.
- **Battle-Tested Heuristics:** The library implements state-of-the-art Local Search and Metaheuristic operators (e.g., LNS, GLS) that have been refined over years of industrial use. Leveraging these built-in mechanics allows the research to focus on *modelling* the complex business constraints rather than re-inventing the wheel of standard VRP moves (e.g., 2-opt implementation details).
- **Proven Scalability:** As demonstrated in the implementation, the library is designed handling complex side constraints without the fragility of pure custom heuristics.

5.1.2. The CP-SAT Exploratory Phase

Before adopting the specialized Routing Library, a generic Constraint Programming (CP) solver (CP-SAT) was evaluated. This phase was critical to validate the hypothesis that a pure constraint programming approach, while expressive, lacks the scalability for routing problems.

Initially, the problem was modeled using the CP-SAT (Constraint Programming - SATisfiability) solver. The primary motivation for this choice was the requirement to model the Minimum Call Size (MCS) and Minimum Overtime Call Size (MOCS) as hard constraints. In CP-SAT, logical implications such as 'If a vessel visits terminal l, it must move at least X containers' can be naturally expressed using boolean

implication logic without linearization artifacts.

5.1.3. The Scalability Bottleneck

While CP-SAT successfully solved small instances, it encountered severe scalability issues with realistic datasets. The core issue was the quadratic growth of the search space. A typical problem instance involves approximately 200 aggregated orders, resulting in at least 400 nodes. In a constraint programming context, this implies a potential search over a fully connected graph:

$$\text{Arcs} \approx N^2 = 400^2 = 160,000 \text{ variables}$$

To mitigate this 'Arc Explosion', we conducted a statistical analysis of the network topology to identify pruning opportunities. This analysis was performed on a hinterland operator's dataset, which contains a large number of closely-spaced inland terminals and thus represents the most challenging case for graph pruning. As shown in the cutoff simulation (Table 5.1), the network exhibits extremely high spatial clustering.

Cutoff Threshold	Arcs Retained	Reduction
50 km	27.2%	72.8%
100 km	41.2%	58.8%
250 km	92.0%	8.0%
500 km	97.7%	2.3%

Table 5.1: Impact of distance thresholds on graph density.

This analysis revealed a critical 'dense graph' problem:

- **Ineffective Pruning:** A conservative cutoff (e.g., 250 km) retains 92% of the arcs, failing to significantly reduce the N^2 complexity space.
- **Network Disconnection:** To achieve a meaningful reduction (e.g., removing 50% of arcs), the threshold must be lowered to ≈ 100 km. However, this removes essential 'long-haul' legs (e.g., Rotterdam to Antwerp) that are required for connectivity, rendering the problem infeasible.

We also attempted K-Nearest Neighbours (KNN), keeping only the K closest neighbours for each node. This failed due to the high density of local clusters (41% of connections are < 100 km) A task in a major terminal often has hundreds of neighbours within a few kilometers. Standard KNN saturated the neighbor list with these local tasks, effectively 'trapping' the vessel in a local cluster and blinding it to distant terminals required for valid routes.

Consequently, the architecture was pivoted to the Google OR-Tools Routing Library. Unlike CP-SAT, which searches for exact proofs of optimality, the Routing Library uses Local Search metaheuristics on top of a constraint programming core. It implicitly handles the N^2 complexity more efficiently by exploring neighbourhood moves (e.g., relocations, swaps).

5.2. System Architecture

To manage the complexity of the data transformation and optimization process, the software is structured as a linear pipeline. As illustrated in Figure 5.1, the system creates a clear separation of concerns between data validation, modelling (problem construction), and solving.

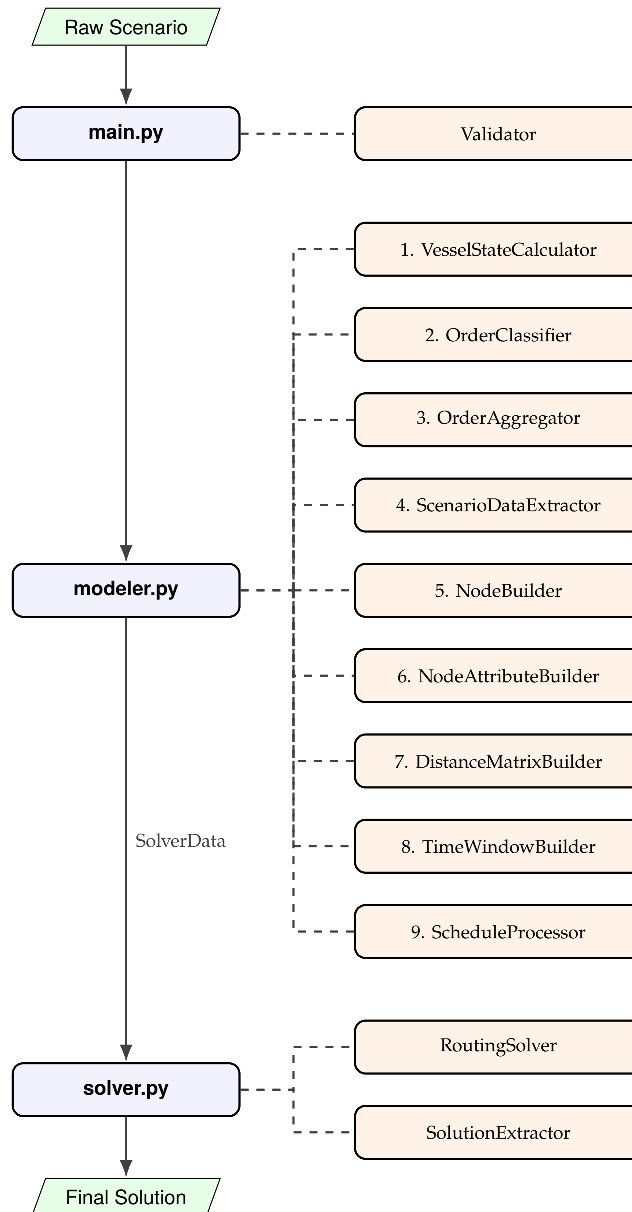


Figure 5.1: Software architecture of the Orion solver.

The `modeler.py` module acts as the central transformation engine. It converts the raw business objects (JSON) into a normalized `SolverData` structure. This design isolates the optimization logic from the data ingestion logic, as demonstrated during development when pivoting from CP-SAT to the Routing Library without rewriting the data parsing code.

5.2.1. The Modeler Pipeline

The data transformation occurs in nine sequential steps, in a fixed dependency order that ensures modularity and maintainability:

1. **VesselStateCalculator:** Determines the VRP starting state for each vessel. It identifies the initial location, time availability, and crucially, the initial cargo on board. It filters out fixed stops that occur after the planning horizon to prevent the solver from optimizing against irrelevant future constraints.
2. **OrderClassifier:** Segments the complete order set into three categories: 'On-board' (already loaded, must be delivered), 'Unplanned' (new orders requiring a full pickup and delivery plan), and 'Fixed' (already part of a historical or locked schedule).
3. **OrderAggregator:** Reduces the problem size by aggregating small, identical orders into larger tasks. As described in Section 5.3, this step enforces a critical capacity constraint: no aggregated group may exceed the capacity of the fleet's smallest vessel.
4. **NodeBuilder:** Creates the master list of all graph nodes. It translates abstract business entities into concrete solver nodes (Pickup, Delivery, On-board Delivery, Fixed Stop, and Depots).
5. **ScenarioDataExtractor:** Extracts and scales simple simulation parameters (e.g., vessel speeds, capacities) from the raw scenario object into solver-ready arrays.
6. **NodeAttributeBuilder:** Calculates the physical demands (TEU, weight) and service times for each node. It computes handling times based on volume (e.g., $Time$ = Volume×HandlingTimePerTEU).
7. **DistanceMatrixBuilder:** Constructs the $N \times N$ travel distance matrix. It maps node indices to terminal IDs and retrieves precise waterway distances from the database, converting kilometers to meters for integer precision.
8. **TimeWindowBuilder:** Converts absolute datetime strings (e.g., '2023-10-27T10:00Z') into relative second-based intervals (e.g., [7200, 14400]) measured from the start of the planning horizon.
9. **ScheduleProcessor:** 'Unrolls' recurring weekly schedules (e.g., 'Open Mondays 09:00-17:00') into concrete, second-based valid intervals across the entire planning horizon, accounting for weekends and night closures.

The output of this pipeline is the `SolverData` object, a normalized, read-only data structure that contains all matrices and arrays required by the `RoutingSolver`.

5.3. Data Reduction and Solver Mapping

While the pipeline described in Section 5.2 outlines the full data transformation process, two specific steps, Order Aggregation and Index Mapping, require detailed elaboration as they are the primary drivers of the solver's performance and scalability.

5.3.1. Order Aggregation Strategy

To mitigate the state-space explosion common in VRPPDTW, the `OrderAggregator` implements a heuristic grouping strategy. As illustrated in Figure 5.2, the raw problem graph (Left) connects every pickup and delivery task individually, resulting in a dense mesh of potential travel arcs ($O(N^2)$).

The graph transformation is achieved by consolidating orders that share identical characteristics: (1) Origin, (2) Destination, (3) Time Window, and (4) Container Type. In the aggregated graph (Right), individual tasks are replaced by 'super-nodes' (e.g., P_A represents the consolidated pickup tasks at Terminal A).

This step is critical for two reasons:

- **Search Space Reduction:** For a representative dataset, this process reduced the order count from 4,868 raw orders to 520 aggregated orders. This $\approx 90\%$ reduction in node count reduces the number of edges in a fully connected graph from $\sim$ 23 million (4868^2) to $\sim 270,000$ (520^2). In practice, the actual arc count is lower due to additional constraints such as forbidden terminals.
- **Feasibility Guarantee:** A critical check ensures that a group of aggregated orders never exceeds the capacity of the smallest vessel in the fleet. This guarantees that any task node in the graph can theoretically be served by any vessel, preventing artificial infeasibility. We note that this check is bypassed for 'On-board Orders' (see Section 5.5.1) since they are physically bound to the specific vessel already carrying them, regardless of the generic fleet minimums.

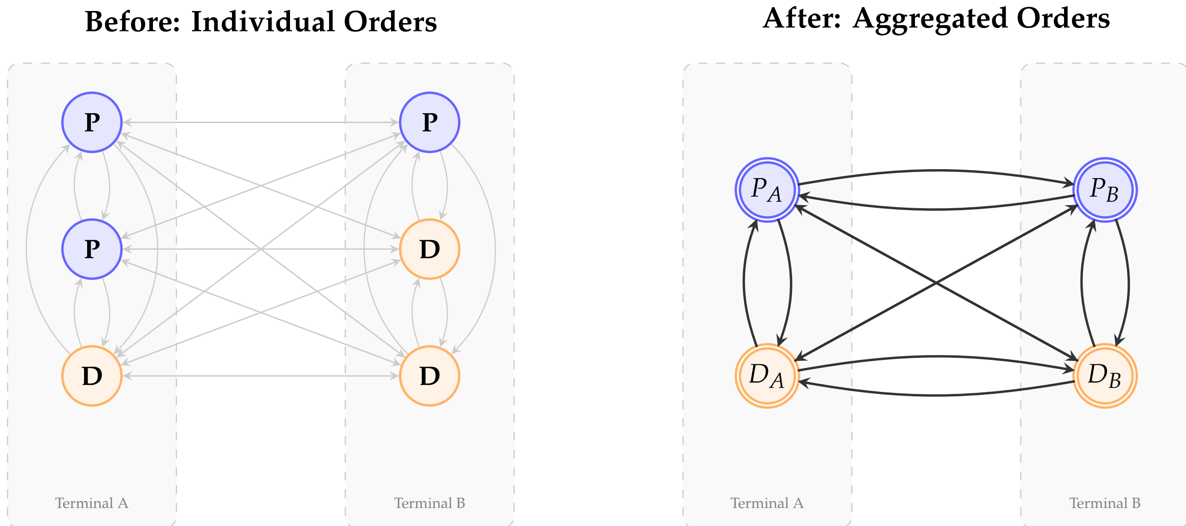


Figure 5.2: Impact of order aggregation on graph complexity. **Left:** Individual orders are represented as distinct nodes. P denotes a Pickup and D denotes a Delivery. The arrows represent the fully connected search space ($N = 6$, Arcs= 30). **Right:** Orders sharing the same origin and destination are grouped into super-nodes. For example, P_A represents all pickups at Terminal A destined for Terminal B. This aggregation reduces the graph size to $N' = 4$ and the search space to 12 arcs.

5.3.2. Index Mapping

The `RoutingSolver` operates on a continuous range of integers (indices), whereas the business logic operates on object IDs. The `RoutingIndexManager` handles this translation.

Each aggregated order $o \in O$ corresponds to a Pickup index and a Delivery index. We explicitly enforce their relationship using the solver's `AddPickupAndDelivery` method, which creates a hard precedence constraint ($Pickup \rightarrow Delivery$) and ensures both are visited by the same vehicle.

5.3.3. Cost Scaling

The Routing Library's objective function relies on integer arithmetic. To prevent precision loss when dealing with currency (cents) or fractional distances, all cost coefficients are scaled by a factor of 100. This implies that a cost of 1 unit in the solver corresponds to 0.01 currency units in reality. This scaling is applied uniformly to distance costs, penalties, and fixed costs.

5.4. Implementing Core Dimensions

In the Routing Library, constraints that accumulate along a route (time, capacity, cost) are modeled as Dimensions.

5.4.1. Time and Variable Speed

Time is modeled as a dimension accumulating seconds. A unique challenge in this implementation is that vessels have different sailing speeds. We cannot use a static time matrix. Instead, we register a transit callback that dynamically calculates travel time based on the specific vehicle assigned to the arc.

The cost of traversing an arc (i, j) includes travel time, handling time at node i, and the berth time if node j represents a new terminal visit. To optimize performance, the components of this calculation are precomputed into arrays during initialization to avoid expensive object lookups. The full implementation of the time callback is provided in Listing A.1 in Appendix A.

5.4.2. Capacity Constraints

Capacity is enforced across four distinct dimensions: `TEU`, `Weight`, `Reefer`, and `DangerGoods`. Unlike the Time dimension, which accumulates value along edges, capacity dimensions represent the instantaneous load on the vessel.

We implement this using `AddDimensionWithVehicleCapacity`. A unary transit callback is registered for each dimension. The capacity dimension construction is detailed in Listing A.2 in Appendix A.

The `fix_start_cumul_to_zero` parameter is set to `False` to support scenarios where vessels begin the schedule with cargo already on board, a common occurrence in rolling-horizon planning.

5.4.3. Charter (Day) Costs

A significant component of the objective function is the Charter Cost, which is a fixed fee incurred for every day a vessel is in use. Properly modelling this cost is challenging because the Routing Library does not natively support a ‘fixed cost per time interval’ structure.

We evaluated three distinct modelling approaches to address this:

1. **Linear Time Cost**: Approximating the daily charter rate as a cost-per-minute penalty.
2. **Explicit Activation Cost**: Applying a fixed penalty to the vehicle if it is used at all, scaled by the planning horizon length.
3. **Implicit (Distance Minimization)**: Ignoring explicit charter costs, focusing exclusively on minimizing travel distance.

Two factors motivated the choice of strategy. First, during empirical testing, explicitly penalizing fleet usage (Strategies 1 and 2) resulted in solutions with higher total costs compared to the Implicit strategy. Second, inconsistent data on vessel ownership (owned vs. chartered) across operators made it impossible to assign meaningful daily cost values. The Implicit approach, which focuses exclusively on minimizing travel distance, consistently produced the best solutions in our benchmarks.

Consequently, the final implementation uses the Implicit strategy (see Listing A.3 in Appendix A).

5.4.4. Modal Shift and Unplanned Orders

A key requirement of the system is the ability to evaluate ‘Unplanned Orders’, which are new transport requests that are not yet committed to the vessel schedule. The solver must determine whether it is more efficient to serve these orders with the vessel fleet or to reject them (implying they will be transported by an alternative mode such as truck or train).

We model this decision using `Disjunctions` with a penalty cost.

For every unplanned order, we assign a penalty value equal to the estimated cost of the cheapest alternative transport mode (e.g., truck or train) for that specific container. We then wrap the Pickup and Delivery nodes in a disjunction:

$$\text{Objective} = \text{RoutingCost} + (\text{IsDropped} \times \text{TruckingCost})$$

The solver’s minimization objective creates a natural economic threshold:

- If the marginal cost of adding the order to a vessel route is *lower* than the alternative cost, the solver will include the order.
- If serving the order requires an expensive detour that exceeds the alternative cost, the solver will ‘drop’ the node, effectively recommending a modal shift to the cheaper alternative (truck or train).

This trade-off is implemented using disjunctions as shown in Listing A.4 in Appendix A.[1]

5.5. Implementing Complex Business Rules

The most significant engineering challenge involved modelling the terminal-level constraints and mandatory breaks within the Routing Library.

5.5.1. Binding Onboard Cargo

A defining characteristic of the 'rolling horizon' planning problem is that vessels rarely start empty. At the beginning of the planning period ($T = 0$), vessels often have cargo on board ('On-board Orders') that must be delivered to specific destinations.

Unlike new orders, which the solver is free to assign to any compatible vessel, on-board orders are physically locked to the vessel currently carrying them. Transferring this cargo to another vessel (transshipment) is not supported in the current operational model. Therefore, we must enforce a strict binding constraint: the delivery node for an on-board order must be visited by the specific vessel carrying that order.

We implement this using the `SetAllowedVehiclesForIndex` method, as shown in Listing A.5 in Appendix A. This acts as a hard constraint on the `VehicleVar` of the node, reducing its domain to a singleton set containing only the index of the carrying vessel.

This constraint not only ensures physical feasibility but also aids the solver by reducing the search space for these specific nodes.

5.5.2. Minimum Call Size: The Penalty Gradient

As noted in the design rationale, modelling MCS as a hard constraint caused infeasibility in the Routing Library. Therefore, we implemented it as a soft constraint with a dynamic penalty.

A naive implementation using a step function (e.g., '0 cost if ≥ MCS, else High Cost') creates a 'plateau' in the search space. If a vessel is visiting a terminal with 5 containers but the MCS is 20, adding one more container (total 6) does not reduce the penalty. The local search heuristic sees no gradient to follow and often stagnates in a local optimum.

To solve this, we implemented a linear penalty gradient (see Listing A.6 in Appendix A). We calculate the deficit and apply a unit cost, creating a continuous slope that guides the solver toward the threshold.

This formulation ensures that every container added to an under-utilized visit reduces the objective function, encouraging the solver to satisfy the constraint incrementally.

Note on Experimental Configuration: While this gradient penalty was fully implemented and tested, final experimental runs revealed that the solver naturally achieved efficient clustering (minimizing travel distance) without explicit MCS enforcement. In fact, disabling this constraint improved solving speed and solution quality for certain dense scenarios, as the 'natural' optimization of travel distance often aligns with visiting terminals only when necessary (i.e., with sufficient cargo). Thus, for the final evaluation, the MCS constraint was disabled.

5.5.3. Minimum Overtime Call Size (MOCS)

The Overtime constraint (MOCS) is stricter: if a vessel visits during 'overtime' hours, it must move a minimum volume. This cannot be easily smoothed with penalties. The implementation relies on boolean logic variables to detect if a visit occurs strictly within overtime.

We define 'Effective Overtime' as the intersection of being inside an overtime window and not inside a regular window. The detection logic is provided in Listing A.7 in Appendix A.

We then enforce the constraint: $\texttt{is_using_overtime} \implies \texttt{total_overtime_volume} \geq \texttt{threshold}$.

[1] While the disjunction mechanism supports realistic per-order trucking costs, the evaluation in Chapter 6 uses a uniformly high penalty for all orders to ensure that both Orion and Baseline SA are compared on identical, fully assigned order sets. This simplification guarantees that neither solver gains an advantage by selectively dropping orders, enabling a fair head-to-head comparison on travel cost alone.

5.5.4. Mandatory Break Intervals

Crew schedules impose inactive windows where vessels cannot travel or service nodes (e.g., night closures from 23:00 to 06:00). We enforce this using the `SetBreakIntervalsOfVehicle` API.

It is critical that the solver understands not only when a break occurs, but also that a break cannot interrupt an atomic operation (like loading a container). The implementation of mandatory breaks is shown in Listing A.8 in Appendix A.

This ensures that if a task takes 30 minutes, the solver will not schedule it to start 15 minutes before a mandatory break.

5.5.5. Forbidden Terminals

Planners restrict certain vessels from visiting specific terminals to enforce fleet segmentation (e.g., keeping vessels within designated regions). This is handled via domain reduction (Listing A.9 in Appendix A). We identify the set of forbidden nodes for each vessel and explicitly remove that vessel index from the node's `VehicleVar` domain.

This approach is more efficient than adding a 'soft penalty' constraint because it prunes the search space at the root, preventing the solver from ever considering invalid assignments.

5.5.6. Terminal Opening Hours

Modelling terminal opening hours requires strict enforcement of when a vessel can dock and perform operations. We adopted a domain reduction strategy using the `RemoveInterval` API.

Our operational profile typically consists of large, continuous availability blocks (e.g., 06:00 to 23:00) separated by substantial overnight closures. This allows us to strictly enforce opening hours by iteratively removing these forbidden 'closed' intervals from the domain of the time variable (see Listing A.10 in Appendix A).

This approach provides an optimal balance of correctness and performance. By natively removing invalid values from the variable domains, the constraint engine can propagate bounds in $O(1)$ time without managing auxiliary boolean variables. This robustly handles the multi-day scheduling constraints required by the business logic.

5.6. Search Configuration

The solver implementation exposes the standard `FirstSolutionStrategy` and `LocalSearchMetaheuristic` enums provided by the OR-Tools library. This design enables the extensive experimental evaluation detailed in Chapter 6, where various constructive heuristics (e.g., `PATH_CHEAPEST_ARC`, `GLOBAL_CHEAPEST_ARC`) and metaheuristics (e.g., `GUIDED_LOCAL_SEARCH`, `TABU_SEARCH`) are benchmarked to identify the optimal configuration for the specific constraints of this problem.

By decoupling the search configuration from the constraint logic, we can empirically evaluate the trade-offs between solution quality, convergence speed, and stability without modifying the core codebase.

5.6.1. Automated Diagnosis Tool

Given the complex interaction of the VRPPDTW constraints, distinguishing between implementation bugs and genuine data infeasibility proved difficult. To address this, we implemented an automated diagnostic tool based on the Delta Debugging (DDMIN) algorithm [14], which isolates the minimal subset of orders causing infeasibility. The theoretical background and specific implementation of this tool are detailed in Appendix B.

5.7. Baseline Solvers

The company maintains an existing automated planning system that predates Orion. This system is implemented in Java and follows the same three-phase architecture common to metaheuristic solvers: initialization, construction, and improvement. Two variants of the system exist, differing only in their improvement phase: one based on Simulated Annealing (Baseline SA) and one based on a

destroy-and-repair strategy inspired by Large Neighbourhood Search (Baseline LNS). Baseline LNS was the original production solver but has since been replaced by Baseline SA. In particular, Baseline LNS was unable to plan all orders under certain constraint configurations, whereas Baseline SA handles the full range of scenarios currently encountered in production. As the current candidate for production deployment, Baseline SA is the variant evaluated in Chapter 6.

5.7.1. Shared Construction Heuristic

Both variants construct an initial solution using a randomized greedy insertion heuristic. Orders are inserted sequentially, starting with discharge orders (cargo already on board) and then regular pickup-and-delivery orders, by evaluating every feasible insertion position across all routes and selecting the cheapest (conceptually similar to cheapest insertion, though distinguished by the randomized tolerance window and order-splitting logic described below). To introduce controlled diversification, the heuristic does not always select the single best insertion. Instead, it builds a candidate set of all insertions whose cost falls within 25% of the cheapest option and randomly selects one. If an order cannot fit entirely on a single vessel (due to capacity), it is split and the remainder is planned iteratively.

5.7.2. Baseline SA: Simulated Annealing

The first variant, referred to as *Baseline SA*, improves the initial solution using Simulated Annealing. The solution is represented as an index-based encoding where each vessel holds an ordered list of order indices. Three neighbourhood operators are applied with fixed probabilities:

- **Swap** (45%): Exchanges two orders between different vessels.
- **Insert** (30%): Moves an order from one vessel to another.
- **Rearrange** (25%): Repositions an order within the same vessel's route.

Each operator uses a hybrid insertion strategy: with 50% probability it attempts a 'smart' placement near terminals already serving similar locations, and otherwise inserts at a random feasible position.

The temperature schedule is calibrated adaptively per problem instance. Before the main loop, 500 random moves are performed and the average cost increase among worsening moves is used as the initial temperature T_{start}. The temperature is then cooled exponentially toward $T_{\text{end}} = 100$ over a maximum of 10^6 iterations. The search terminates when either the temperature floor is reached, the iteration limit is exhausted, an 8-minute time limit expires, or no improvement has been found for 1 minute.

5.7.3. Baseline LNS: Destroy-and-Repair

The second variant, referred to as *Baseline LNS*, uses a destroy-and-repair strategy inspired by Large Neighbourhood Search. Rather than making small, incremental moves like Baseline SA, it operates at the route level: for each vessel in turn, all stops are removed from the route and the displaced orders are re-inserted using the greedy heuristic. If the resulting solution improves upon the current best, it is accepted; otherwise, it is discarded.

This process runs in two phases:

1. **Diversification phase:** The greedy re-insertion uses a 10% cost tolerance (candidate set within 10% of the cheapest insertion), allowing some exploration. This phase iterates until a full pass through all routes yields no improvement.
2. **Intensification phase:** The tolerance is reduced to 0%, forcing purely greedy insertions. This phase also iterates until convergence.

Unlike Baseline SA, Baseline LNS is strictly descent-based: it never accepts a worsening solution. This makes it faster to converge but more susceptible to local optima, as it cannot escape poor basins of attraction through uphill moves.

As noted above, Baseline LNS has been replaced by Baseline SA as the production solver. It is therefore not included in the experimental evaluation (Chapter 6).

6

Evaluation

This chapter presents the experimental evaluation of the Orion solver. The evaluation is structured in three phases:

1. **Feasibility Analysis:** Filtering the 14 considered construction heuristics to identify viable candidates.
2. **Parameter Tuning:** Evaluating the selected strategies and metaheuristics to identify the best-performing configuration.
3. **Benchmarking:** Comparing the optimized Orion solver against human planners and the company's Baseline SA.

6.1. Experimental Setup

6.1.1. Environment

All experiments were conducted on a Lenovo ThinkPad running Windows 11 Pro with an Intel Core i7-1165G7 processor (4 cores, 8 threads, base frequency 1.2 GHz, turbo up to 4.7 GHz) and 48 GB of RAM. The Orion solver was implemented in Python 3.12 using Google OR-Tools v9.10. To ensure a fair comparison, both Orion and the baseline solvers were executed on the same hardware under identical conditions.

6.1.2. Datasets

The experiments use six real-world planning scenarios sourced from three of the company's operator clients (Operators A, B and C, as described in Chapter 2). Each scenario represents a single planning session: a snapshot of the fleet state, pending orders, and onboard cargo at a specific date. Table 6.1 summarizes the key characteristics of each scenario. The scenarios vary considerably in scale (25 to 184 aggregated orders), fleet size (4 to 13 vessels), network density (5 to 55 terminals), and planning horizon (7 to 27 days), providing a diverse evaluation set for the solver.

Table 6.1: Data profiles for the six evaluation scenarios. Order counts in parentheses denote the number of aggregated orders passed to the solver.

Scenario	Operator	Vessels	Unplanned	Onboard	Terminals	Horizon
August 4	B	7	2446 (54)	554 (6)	5	8 days
April 14	B	7	2799 (63)	172 (1)	6	8 days
March 23	A	13	1202 (136)	544 (48)	40	14 days
October 27	A	4	927 (129)	268 (22)	17	14 days
April 28	C	10	18 (3)	142 (22)	22	27 days
March 30	A	13	599 (67)	783 (86)	55	7 days

Note on Order Aggregation. As described in Section 5.3, individual container orders sharing the same origin, destination, and time window are aggregated into compound orders before being passed to the solver. The "Unplanned" and "Onboard" columns in Table 6.1 report both the raw order count and the aggregated count (in parentheses). The aggregated count determines the number of pickup–delivery node pairs in the routing model and thus directly governs solver complexity.

Note on Scenario Diversity. The April 28 scenario (Operator C) is notably different from the others: it features only 25 aggregated orders across 10 vessels but spans a 27-day horizon with 22 terminals. This represents a short-distance inland waterway operation where vessels shuttle between a small number of terminals. As we will see, both Orion and the baseline solvers converge almost instantly on this input.

6.1.3. Key Performance Indicators

The primary metric for evaluation is the Travel Cost, defined as the total distance travelled by all vessels multiplied by their per-kilometre cost rate (i.e., $\sum_v C_v^{Trans}$ from the objective function in Chapter 4).

As discussed in Section 5.3, charter costs (C_v^{Usage}) are treated as sunk for owned vessels, and handling costs are constant across solutions for a fixed order set. Therefore, the effective optimization objective reduces to minimizing travel cost, subject to the constraint that all orders must be assigned.

Secondary metrics include:

- **Unassigned Orders:** The number of orders the solver failed to insert. A solution with unassigned orders is considered *infeasible* for benchmarking purposes, as it indicates the solver could not satisfy all hard constraints within the time limit.
- **Convergence Profile:** The percentage improvement in objective value achieved within successive time intervals (0–30s, 30–60s, Min 2, . . . , Min 5). This reveals how quickly the solver reaches diminishing returns.
- **Number of Solutions Found:** The total number of improving solutions discovered during the search, which indicates how actively the metaheuristic explores the search space.

6.1.4. Baselines

The Orion solver is compared against two baselines:

1. **Human Planners:** The actual plans produced by each operator's logistics team. These serve as the real-world reference point. Table 6.2 reports the travel cost of each human plan.
2. **Baseline SA:** The Simulated Annealing variant of the company's existing automated planning system, described in detail in Section 5.7. Recall that it uses a randomized greedy construction heuristic followed by SA-based improvement with three neighbourhood operators (swap, insert, rearrange). Baseline SA was run four times per scenario with an iteration count configured to approximate a 5-minute runtime (actual runtimes were at most 5 minutes), matching the operational time budget. As the current candidate for production deployment, it represents the strongest available automated baseline.

Table 6.2: Travel cost of human-generated plans for each scenario.

Scenario	Human Plan Travel Cost
August 4	124,406
April 14	101,529
March 23	28,103
October 27	16,313
April 28	188,116
March 30	26,220

Note on Order Parity. To ensure a fair comparison between Orion and Baseline SA, we validated that both solvers operate on the exact same set of orders for each scenario. A verification script extracted all order IDs across all routes and stops in each solution and confirmed that the order sets are identical.

The same script verified that both solvers use the same distance matrix, ensuring no model gains an unfair advantage through data discrepancies.

6.2. Phase 0: Feasibility Analysis

Before tuning, we conducted a feasibility study to filter the 14 First Solution Strategies provided by OR-Tools. A strategy was considered *feasible* if it could produce a valid solution with zero unassigned orders within a 60-second timeout.

Results. Out of 14 strategies, only two successfully found a complete solution for the March 23 scenario (selected as the stress test due to its high terminal count and order volume) within the time limit:

- `PARALLEL_CHEAPEST_INSERTION` (PCI)
- `LOCAL_CHEAPEST_INSERTION` (LCI)

Other common strategies such as `PATH_CHEAPEST_ARC` and `GLOBAL_CHEAPEST_ARC` either timed out (exceeded 60 seconds) or failed to insert all orders. This is consistent with the literature, where cheapest insertion heuristics tend to perform well on tightly constrained pickup-and-delivery problems because they evaluate the cost impact of each insertion, whereas nearest-neighbor–based strategies greedily extend routes without considering downstream feasibility.

Consequently, PCI and LCI were selected as the primary candidates for the tuning phase.

6.3. Phase 1: Parameter Tuning

In this phase, we evaluated all feasible solver configurations by crossing the two construction heuristics (PCI, LCI) with five local search metaheuristics provided by OR-Tools:

- Guided Local Search (GLS)
- Tabu Search (TS)
- Generic Tabu Search (GTS)
- Simulated Annealing (SA)
- Greedy Descent (GD)

This yields $2 \times 5 = 10$ configurations. Each configuration was run twice on each of the six input scenarios for 5 minutes per run, totaling 120 runs. Full results for all runs are reported in Appendix C (Table C.1).

6.3.1. Ranking by ARPD

To identify the best configuration, we use the Average Relative Percentage Deviation (ARPD) from the best-known solution. For each scenario i and configuration c, the RPD is defined as:

$$RPD_{i,c} = \frac{f_{i,c} - f_i^*}{f_i^*} \times 100\% \tag{6.1}$$

where $f_{i,c}$ is the best final objective value achieved by configuration c on scenario i, and f_i^* is the best objective value achieved by *any* configuration on scenario i. The ARPD for a configuration is then the average RPD across all scenarios.

Table 6.3 presents the ranking.

6.3.2. Observations

LCI dominates PCI. All five LCI-based configurations outperform their PCI counterparts. The data in Appendix C supports this: LCI consistently produces initial solutions with lower objective values than PCI, giving the metaheuristic a better starting point. For example, on the August 4 scenario, LCI's initial objective is 2,045,614 compared to PCI's 9,150,265. While both are far from optimal (due to high penalties for initially infeasible intermediate states), the LCI starting point allows the local search to reach competitive solutions faster.

Table 6.3: Configuration ranking by Average Relative Percentage Deviation (ARPD). Lower is better.

Rank	Configuration	ARPD
1	LCI + TS	2.72%
2	LCI + GLS	5.94%
3	LCI + GD	9.68%
4	LCI + GTS	10.87%
5	LCI + SA	11.44%
6	PCI + TS	13.54%
7	PCI + GLS	17.64%
8	PCI + SA	20.40%
9	PCI + GD	22.49%
10	PCI + GTS	22.92%

On the more constrained scenarios (August 4, April 14), PCI-based configurations with weaker metaheuristics (GD, SA, GTS) frequently fail to eliminate all unassigned orders within the 5-minute window, resulting in infeasible solutions with inflated objective values. This pattern does not occur with LCI, confirming that the quality of the initial solution is critical for these highly constrained instances.

Tabu Search achieves the best refinement. Among the metaheuristics, Tabu Search consistently produces the lowest final objectives. This can be attributed to its memory mechanism: by maintaining a tabu list of recently visited solutions, TS avoids cycling and explores a broader neighbourhood than greedy or purely stochastic methods. GLS ranks second, leveraging its penalty-based landscape augmentation to escape local optima, though it tends to plateau slightly earlier than TS on larger instances.

Convergence is rapid. Across all configurations and scenarios, the majority of improvement occurs within the first 30 seconds. The "0–30s" column in Table C.1 shows improvements of 96–99.97% of the total gain, depending on the scenario. After the first minute, further improvements are marginal, typically below 5% of the remaining gap. An important caveat applies to these percentages: the objective value after the construction heuristic is typically very high, because any unassigned order incurs a large penalty. The first seconds of local search are therefore dominated by fitting these remaining orders into the schedule, which produces dramatic drops in objective value but does not yet represent route-quality improvement. The subsequent fine-tuning phase, where the solver reorganizes routes to reduce actual travel cost, accounts for a small fraction of the total objective improvement but can make a meaningful difference in practice. The notable exception is the April 28 scenario (Operator C), where all configurations converge in under 30 seconds due to the small problem size (25 aggregated orders).

Results are highly deterministic. For most configurations, the two runs produce identical final objective values. This is expected for deterministic metaheuristics (GD, GTS) but is also observed for Tabu Search and GLS, which have stochastic elements. The determinism arises because OR-Tools' local search operators are applied in a fixed order, and with identical starting solutions (same construction heuristic, same data), the search trajectory is reproducible. This is a practical advantage: operators do not need to run the solver multiple times to verify the result. However, determinism is a double-edged sword: if the solver converges to a suboptimal local optimum, it will consistently return that same result. In contrast, Baseline SA's stochastic restarts occasionally discover better solutions (as observed on April 14), precisely because different random seeds explore different regions of the search space. Thus, while determinism eliminates variance, it also eliminates the chance of a lucky escape from a poor basin of attraction.

6.3.3. Local Search Operator Activity

To understand *how* the solver improves solutions, we analyzed the local search operator logs across all 120 Phase 1 runs (5 minutes each). Table 6.4 reports the total number of accepted moves per operator, aggregated across all scenarios and configurations.

Table 6.4: Accepted moves by local search operator across all Phase 1 runs.

Operator	Moves	Share
RelocateNeighbors	7,732	20.7%
OrOpt (chain length 1)	7,632	20.4%
OrOpt (chain length 2)	6,898	18.5%
OrOpt (chain length 3)	4,296	11.5%
Relocate	4,028	10.8%
RelocateExpensiveChain	2,993	8.0%
PairRelocate	1,578	4.2%
RelocateSubtrip	616	1.7%
TwoOpt	332	0.9%
MakeInactive	321	0.9%
MakeActive	321	0.9%
Cross	310	0.8%
Exchange	147	0.4%
ExchangeSubtrip	42	0.1%
HeuristicPathLNS (Global)	28	<0.1%
HeuristicPathLNS (Local)	23	<0.1%
PairExchange	14	<0.1%
MakePairActive	7	<0.1%
MakePairInactive	3	<0.1%
Total	**37,321**	**100%**

Relocate-family operators dominate. The top seven operators are all variants of the Relocate move described in Section 3.2.3: they move a node, chain, or pickup-delivery pair from one position to another. Together, they account for over 95% of all accepted moves. OrOpt, which relocates chains of length 1, 2, or 3, is the single most active operator family (50.4% combined). This aligns with the nature of our problem: because orders are aggregated into compound pickup-delivery pairs, the solver primarily improves solutions by repositioning these pairs and their surrounding chains rather than by swapping or reversing segments.

LNS operators fire rarely. Despite the theoretical importance of Large Neighbourhood Search (Section 3.2.3), the filtered heuristic LNS operators account for only 51 accepted moves (0.14% of the total). This suggests that for our problem instances and time budgets, the bulk of improvement comes from fast, fine-grained neighbourhood moves rather than large-scale destroy-and-repair operations. It also partially explains the rapid convergence observed earlier: the solver reaches diminishing returns quickly because the dominant operators perform small, incremental improvements.

Swap and 2-Opt are secondary. The Exchange (Swap) and TwoOpt operators, which are considered core operators in classical VRP literature, contribute only 479 accepted moves combined (1.3%). This is consistent with the pickup-and-delivery structure of the problem: reversing a sub-chain (2-Opt) or exchanging two nodes (Swap) is more likely to violate precedence constraints (pickup before delivery), making these moves less frequently feasible compared to targeted relocations.

6.3.4. Selected Configurations

Based on the ARPD ranking, we select the top two configurations for extended evaluation in Phase 2:

1. **LCI + TS** (ARPD: 2.72%), the best overall configuration
2. **LCI + GLS** (ARPD: 5.94%), the second best, useful for comparison

6.4. Phase 2: Benchmark Comparison

This phase compares the optimized Orion solver (using LCI + TS and LCI + GLS) against the two baselines: human planners and Baseline SA.

6.4.1. Extended Orion Runs

To verify that the 5-minute time budget is sufficient, the two best configurations were run for 10 minutes on each scenario. Since Phase 1 demonstrated high determinism, a single run per configuration was deemed sufficient. Full results are reported in Table C.2 (Appendix C).

Results. The 10-minute runs confirm that extending the time budget provides negligible benefit. For LCI + TS, the final objective is identical to the 5-minute result on five of six scenarios. The only exception is the March 23 scenario, where the 10-minute run achieves a marginally different travel cost (23,378 vs. 24,142), with improvement occurring after minute 3. For LCI + GLS, one scenario (April 14) shows a small improvement (108,072 vs. 108,841), attributed to a single improving move in minute 7 of the run.

These results validate the 5-minute operational time budget: the solver effectively plateaus within the first 1–2 minutes, and extending to 10 minutes yields at most 3% additional improvement on a single scenario.

6.4.2. Baseline SA Results

Baseline SA was run four times per scenario for 5 minutes each. Table 6.5 reports the travel cost for each run.

Table 6.5: Baseline SA travel cost across four runs per scenario (5-minute time limit).

Scenario	Run 1	Run 2	Run 3	Run 4	Best	Avg
August 4	126,219	120,927	132,357	124,753	120,927	126,064
April 14	83,113	97,978	98,507	94,326	83,113	93,481
March 23	30,053	29,679	29,136	33,545	29,136	30,603
October 27	15,073	17,317	20,732	20,732	15,073	18,464
April 28	132,627	132,627	169,626	132,627	132,627	141,877
March 30	23,747	23,661	23,366	24,311	23,366	23,771

Variance. A notable characteristic of Baseline SA is the variance across runs. The implementation uses a different random seed for each run, which affects both the randomized greedy construction and the SA acceptance decisions, causing different runs to converge to different local optima. While the seed could in principle be fixed to achieve deterministic behaviour, the current production configuration does not do so. This is most pronounced on the April 14 scenario, where the worst run (98,507) is 18.5% more expensive than the best (83,113), and on the October 27 scenario, where the worst run (20,732) is 37.5% more expensive than the best (15,073). On the April 28 scenario, three of four runs converge to the same value (132,627), with one outlier at 169,626, likely because the annealing process got trapped in a poor region of the search space early on.

This variance has practical implications: under the current configuration, operators may need to run the solver multiple times and select the best result, effectively multiplying the computational budget.

6.4.3. Extended Baseline SA Runs

To mirror the extended Orion evaluation in Section 6.4.1, Baseline SA was also run for longer durations on each scenario (two additional runs per scenario). However, because Baseline SA uses iteration-based termination rather than a wall-clock time limit, achieving consistent 10-minute runtimes was not feasible. The actual runtimes ranged from 2 minutes 37 seconds (April 28) to 8 minutes 52 seconds (August 4), depending on the number of iterations configured. Full results, including convergence profiles, are reported in Table C.3 (Appendix C).

Results. Table 6.6 summarizes the travel costs for the extended runs.

Of the six scenarios, the extended runs improved upon the best result from the original four 5-minute runs (Table 6.5) on only two: March 23 (26,326 vs. 29,136, a 9.6% improvement) and March 30 (22,974 vs. 23,366, a 1.7% improvement). On the remaining four scenarios, the extended runs produced results equal to or worse than the original batch. This further underscores the stochastic nature of Baseline SA:

Table 6.6: Extended Baseline SA travel cost and runtimes for two additional runs per scenario.

Scenario	Run 1	Run 2	Best	RT 1	RT 2
August 4	132,363	129,092	129,092	8m 24s	8m 52s
April 14	88,813	87,583	87,583	5m 53s	7m 03s
March 23	28,332	26,326	26,326	8m 16s	8m 17s
October 27	16,162	19,520	16,162	8m 43s	7m 41s
April 28	169,626	132,627	132,627	2m 37s	5m 53s
March 30	24,007	22,974	22,974	4m 57s	4m 42s

additional time and runs do not guarantee improvement, and the search trajectory depends heavily on the random seed and initial configuration.

Convergence. The convergence profiles in Appendix C show that Baseline SA's internal objective, which includes large penalty terms for constraint violations, typically stabilizes within 3–5 minutes. However, unlike Orion, where improvement is concentrated in the first 1–2 minutes, some Baseline SA runs exhibit meaningful cost reductions as late as minute 6–8 (e.g., October 27 Run 1, which shows improvements of 6.97%, 13.85%, and 4.30% in minutes 6, 7, and 8 respectively). This behaviour is explained by the cooling schedule: when the iteration count is increased to extend the runtime, the start and end temperatures remain similar, so the temperature decreases at a slower rate. The search therefore remains explorative for longer, maintaining a higher acceptance probability for worse solutions and enabling escapes from local optima later in the run.

6.4.4. Head-to-Head Comparison

Table 6.7 presents the central comparison between the three approaches: human planners, Baseline SA (best and average of 4 runs), and Orion (LCI + TS, 5-minute run).

Table 6.7: Travel cost comparison across all scenarios. BSA reports the average over 4 runs. Δ columns are relative to the human plan; Δ BSA vs Orion is Orion's cost relative to BSA. Bold indicates the lowest cost per scenario.

Scenario	Human	BSA	Orion	Δ BSA	Δ Orion	Δ BSA vs Orion
August 4	124,406	126,064	**104,826**	+1.3%	**−15.7**%	**−16.8**%
April 14	101,529	**93,481**	104,205	**−7.9**%	+2.6%	+11.5%
March 23	28,103	30,603	**24,142**	+8.9%	**−14.1**%	**−21.1**%
October 27	16,313	18,464	**15,213**	+13.2%	**−6.7**%	**−17.6**%
April 28	188,116	141,877	**132,630**	**−24.6**%	**−29.5**%	**−6.5**%
March 30	26,220	23,771	**17,618**	−9.3%	**−32.8**%	**−25.9**%

6.4.5. Analysis per Scenario

August 4 (Operator B). Orion achieves the lowest travel cost at 104,826, a 15.7% reduction over the human plan and 13.3% below Baseline SA's best run. This is a large-scale short-sea scenario with 7 vessels and 60 aggregated orders. The significant gap over Baseline SA suggests that Orion's local search operators are more effective at reorganizing routes in this scenario.

April 14 (Operator B). This is the only scenario where Baseline SA's best run substantially outperforms Orion (83,113 vs. 104,205). However, this result warrants careful interpretation. Baseline SA's best solution uses 6 vessels, while all other runs (both Baseline SA and Orion) use 7. The idle vessel (which we will call Vessel E) contributes approximately 20,000 in travel cost in Orion's solution. Baseline SA appears to have found a favourable redistribution of Vessel E's workload across the remaining 6 vessels, but achieved this only once in four attempts.

To investigate whether Orion could replicate this fleet reduction, three experiments were conducted:

- **Vehicle fixed-cost penalty:** OR-Tools provides a `SetFixedCostOfAllVehicles` mechanism to penalize each active vessel in the objective. Three penalty levels were tested (10,000, 50,000, and 200,000). In all cases, the solver still activated all 7 vessels. At the 50,000 penalty level, the travel

cost actually worsened to 109,000, likely because the penalty distorted the objective landscape without achieving fleet reduction. The default local search operators (Relocate, Exchange) move 1–3 nodes at a time and cannot coordinate the large sequence of moves needed to empty a vessel.

- **Manual vessel removal:** Vessel E was removed from the input, forcing Orion to solve with 6 vessels. The result was worse: a travel cost of 114,000 with 20 unassigned orders. Without the flexibility to redistribute orders across 7 vessels, the construction heuristic produced a poor starting point from which Tabu Search could not recover.
- **Enabling Full-Path LNS:** OR-Tools provides a `FULL_PATH_LNS` operator (disabled by default, and distinct from the fine-grained LNS operators already active; see Table 6.4) that destroys an entire route and attempts to reinsert its nodes. When explicitly enabled, the solver still activated all 7 vessels and produced a travel cost negligibly different from the default configuration. Although the operator can structurally empty a route, the repair phase appears unable to feasibly redistribute all orders under the tight pickup-and-delivery, time window, and capacity constraints of this instance.

This analysis reveals that both solvers are susceptible to local optima, but the manifestation differs. Orion is stable but gets trapped in a 7-vessel local optimum; even enabling OR-Tools' dedicated full-route-destruction operator (`FULL_PATH_LNS`) does not overcome this. Baseline SA is volatile, usually landing near 94,000–98,000 but occasionally stumbling into a much better 6-vessel configuration. Neither solver reliably finds the globally optimal fleet assignment for this instance.

When comparing Baseline SA's *average* performance (93,481) rather than its best, the gap with Orion narrows to approximately 10%.

March 23 (Operator A). Orion outperforms both baselines, achieving a 14.1% reduction over the human plan. This is the scenario with the most terminals (40) and a 14-day horizon, creating a complex network where Orion finds more efficient routings than both humans and Baseline SA. Baseline SA's best run (29,136) is actually 3.7% worse than the human plan on this input.

October 27 (Operator A). Baseline SA's best run (15,073) narrowly beats Orion (15,213) by 0.9%. However, Baseline SA's average (18,464) is 21.4% higher than its best, whereas Orion is fully deterministic. In an operational setting where the solver is run once, Orion provides a more reliable outcome.

April 28 (Operator C). Both solvers converge to virtually the same solution (Baseline SA: 132,627; Orion: 132,630, a negligible difference of 3). This confirms the expectation from Section 6.1.2: with only 25 aggregated orders, the problem is small enough that any reasonable search strategy will find the same (likely optimal) solution. Both methods dramatically outperform the human plan (−29.5%), suggesting substantial room for routing optimization in this intra-port shuttle operation. However, as discussed in Section 6.5.4, the human planner may have accounted for operational factors (e.g., water levels, berth availability) not captured by either solver, which would partly explain the cost gap.

March 30 (Operator A). Orion achieves the largest improvement of any scenario: a travel cost of 17,618, which is 32.8% below the human plan and 24.6% below Baseline SA's best run. This 7-day scenario combines the highest terminal density (55 terminals) with the largest number of aggregated orders (153) and the shortest horizon, creating many opportunities for the local search to merge and reorder stops across vessels.

6.5. Discussion

6.5.1. Comparative Analysis

The benchmark results paint a nuanced picture. Orion outperforms or matches Baseline SA's best result on four of six scenarios, and outperforms Baseline SA's average on five of six. However, on the April 14 scenario, Baseline SA's best run finds a substantially better solution that Orion cannot replicate.

When considering *reliability*, Orion has a clear advantage. Its deterministic behaviour means that a single run suffices, whereas Baseline SA's stochastic nature requires multiple runs to maximize the chance

of hitting a good local optimum. In an operational context where planners need a schedule within 5 minutes, running the solver four times would require 20 minutes, exceeding the typical replanning window.

Against human planners, both automated approaches deliver substantial improvements. Orion reduces travel cost on five of six scenarios, with reductions ranging from 6.7% to 32.8%. The sole exception is April 14, where the human plan (101,529) narrowly beats Orion (104,205, +2.6%), suggesting that experienced human planners can match or exceed solver performance on familiar routes. That said, as noted in Section 6.5.4, human planners may incorporate operational knowledge (e.g., water levels, berth congestion) that neither solver captures, so the cost differences do not necessarily reflect planning quality alone.

6.5.2. Consistency vs. Peak Performance

A key finding is the trade-off between consistency and peak performance. Table 6.8 quantifies this by comparing the spread between Baseline SA's best and worst runs across the four runs per scenario.

Table 6.8: Baseline SA variance analysis across 4 runs per scenario. Spread = (Worst − Best) / Best.

Scenario	Baseline SA Best	Baseline SA Worst	Spread
August 4	120,927	132,357	8.6%
April 14	83,113	98,507	18.5%
March 23	29,136	33,545	15.1%
October 27	15,073	20,732	37.5%
April 28	132,627	169,626	27.9%
March 30	23,366	24,311	4.0%

Orion's spread is 0% on all scenarios (identical results across runs), making it a more predictable tool for daily operations. In a commercial planning environment, predictability has intrinsic value: planners can trust that the output is the best the solver can achieve, without second-guessing whether re-running might yield a better result.

6.5.3. Advantages of a Solver-Based Approach

Beyond raw KPI comparisons, the solver-based approach offers structural advantages over custom metaheuristics like Baseline SA:

Extensibility. Adding new constraints (e.g., bridge height restrictions, lock scheduling, water level limits) requires only defining new dimension callbacks or domain reductions in OR-Tools, without modifying the search operators. In Baseline SA, adding a new constraint requires implementing penalty logic within the evaluation function and, in some cases, updating feasibility pre-filters. While the temperature schedule auto-calibrates to the new penalty landscape, the process is more tightly coupled to the solver's internal architecture.

Robustness. The solver relies on a battle-tested Constraint Programming core that guarantees hard constraint satisfaction by construction. Custom metaheuristics must implement feasibility checks manually, creating risk of subtle constraint violations.

Reduced Parameter Sensitivity. Orion's performance is governed primarily by the choice of construction heuristic and metaheuristic (a discrete choice), whereas Baseline SA requires careful calibration of continuous parameters (initial temperature, cooling rate, neighbourhood probabilities). As our Baseline SA results show, the use of random seeds in the current configuration leads to high variance across runs.

6.5.4. Limitations

Several limitations should be considered when interpreting these results. The modelling assumptions stated in Section 4.3 (deterministic travel times, static capacity, fixed drop penalty) each have consequences for the evaluation:

Impact of Travel Time Assumptions. Because the solver uses a conservative fixed speed per vessel that intentionally underestimates actual vessel capabilities (see Section 4.3), the resulting travel times are likely overestimated. Consequently, the human plan travel costs reported here may not be directly comparable, as the distance matrix used by the solver may differ from what the human planner assumed. We attempted to validate speeds by computing average vessel speeds from historical data, but the presence of idle time in the data (indistinguishable from slow travel) prevents a reliable comparison.

Impact of Water Level Omission. Neither Orion nor Baseline SA models water level constraints in the current experiments. Future iterations of the solver must dynamically validate the vessel's gross draft against regional water depths at the projected time of transit. In the six scenarios evaluated in this thesis, water level restrictions were not actively binding, so this omission does not affect the current comparison. However, in periods of low water, human planners routinely account for water levels by reducing vessel loads or choosing alternative routes, decisions that increase travel cost but ensure operational feasibility, and this effect would not be captured by the solver.

Impact of Wild Anchorage Assumption. As described in Section 4.3, the model assumes vessels can stop anywhere along a waterway when daily operating hours expire. In reality, inland vessels must dock at designated overnight parking locations. The conservative fixed speeds used in the model are intended to buffer this optimistic assumption, but the resulting schedules may still place vessels at locations where no parking facilities exist. This limitation applies only to inland operators (Operators A and C); the short-sea Operator B operates continuously.

Comparison Fairness. Ideally, both Orion and Baseline SA would represent the most optimized versions of their respective approaches (OR-Tools routing and Simulated Annealing). In practice, neither may be fully optimal: Orion's constraint modelling choices may not be the most efficient possible, and Baseline SA's parameter tuning may not be exhaustive. The results should therefore be interpreted as a comparison between two specific implementations rather than between the theoretical limits of their underlying methodologies.

6.5.5. Threats to Validity

Internal Validity. All experiments were conducted on the same hardware, using the same input data and distance matrices (verified by the order parity check described in Section 6.1.4). The Orion solver has an extensive test suite verifying that all implemented hard constraints (time windows, capacities, forbidden terminals) are never violated.

External Validity. The evaluation covers six scenarios from three operators with distinct operational profiles, providing reasonable diversity. However, the results may not generalize to fundamentally different network topologies (e.g., river-only networks with sequential locks, or deep-sea intercontinental shipping).

Construct Validity. Travel cost is the primary metric, which is standard in VRP literature. However, in practice, operators may value other factors (schedule robustness, buffer time, customer preferences) that are not captured by this single metric.

7 Conclusion and Future Work

7.1. Conclusion

This thesis set out to develop and evaluate an improved scheduling system for the container vessel planning problem. The problem was formulated as a Heterogeneous Vehicle Routing Problem with Pickup and Delivery and Time Windows (VRPPDTW), incorporating domain-specific constraints such as forbidden terminals, terminal opening hours, and mandatory vessel breaks. The resulting solver, Orion, was implemented using the Google OR-Tools Routing Library and evaluated on six real-world planning scenarios from three logistics operators with distinct operational profiles.

We now return to the research questions posed in Chapter 1.

RQ1: How does the solver-based approach compare to the existing Baseline SA and human planners in terms of travel cost? Orion outperforms or matches Baseline SA's best result on four of six scenarios and outperforms Baseline SA's average on five of six. Against human planners, Orion achieves travel cost reductions of 6.7% to 32.8% on five of six scenarios. The largest improvement is observed on the March 30 scenario (Operator A), where Orion reduces travel cost by 32.8% relative to the human plan and 25.9% relative to Baseline SA's average.

The one scenario where Baseline SA substantially outperforms Orion is April 14 (Operator B). Here, Baseline SA's best run discovers a favourable 6-vessel configuration that reduces travel cost to 83,113, compared to Orion's 104,205. However, this result occurred only once in four attempts, with the remaining runs averaging 96,937. Experiments with vehicle fixed-cost penalties, manual vessel removal, and explicit activation of OR-Tools' built-in `FULL_PATH_LNS` operator all failed to achieve fleet reduction, indicating that the problem's tight constraint structure (pickup-and-delivery precedence, narrow time windows, vessel compatibility) prevents even route-destruction operators from feasibly redistributing a vessel's workload.

Across all scenarios, the reported cost figures are subject to the modelling assumptions discussed in Chapter 6: fixed travel speeds, static vessel capacity, and the wild anchorage approximation. Human plan costs in particular may reflect operational considerations not captured by either solver, such as water level constraints or berth availability.

RQ2: What is the impact of different construction heuristics and local search metaheuristics on solver performance? The feasibility analysis (Phase 0) eliminated 12 of 14 available construction heuristics, leaving only Local Cheapest Insertion (LCI) and Parallel Cheapest Insertion (PCI) as viable candidates for the tightly constrained pickup-and-delivery problem. The parameter tuning (Phase 1) revealed two clear findings. First, LCI dominates PCI: all five LCI-based configurations outperform their PCI counterparts, because LCI consistently produces initial solutions with lower objective values, giving the metaheuristic a better starting point. Second, Tabu Search achieves the best refinement among the five metaheuristics, yielding an Average Relative Percentage Deviation (ARPD) of 2.72% compared to 5.94%

for the second-best (Guided Local Search). Tabu Search's memory mechanism, which maintains a tabu list of recently visited solutions, avoids cycling and enables broader neighbourhood exploration.

Convergence analysis showed that 96–99.97% of objective improvement occurs within the first 30 seconds, and extending the time budget from 5 to 10 minutes yields at most 3% additional improvement on a single scenario. However, these percentages are relative to the initial objective value, which is dominated by high penalty costs for unassigned orders. The early improvement largely reflects the solver fitting remaining orders into feasible schedules, while the subsequent fine-tuning of route quality, though small in relative terms, can still be operationally significant.

RQ3: How does the solver-based approach compare to Baseline SA in terms of solution consistency and operational reliability? Orion produces identical results across repeated runs on all scenarios, yielding a coefficient of variation of effectively 0%. In contrast, Baseline SA exhibits substantial variance: the spread between the best and worst run ranges from 4.0% (March 30) to 37.5% (October 27), with coefficients of variation up to 15.2%.

This determinism is a direct consequence of OR-Tools' fixed operator application order combined with the deterministic construction heuristic (LCI in the selected configuration). In an operational context, this means a single Orion run suffices, whereas Baseline SA requires multiple runs to maximize the probability of finding a good solution, effectively multiplying the computational budget.

Summary. The primary contribution of this thesis is a flexible, high-performance optimization model for the intermodal vessel scheduling problem. Orion provides the logistics service provider with a solver that is competitive with or superior to the existing Baseline SA on the majority of tested scenarios, while offering three structural advantages: deterministic output, modular constraint handling through the Constraint Programming paradigm, and reduced parameter sensitivity.

7.2. Recommendations for the Logistics Service Provider

Based on the experimental results, we offer the following recommendations:

1. **Adopt LCI + Tabu Search as the default configuration.** This combination achieved the lowest ARPD (2.72%) across all scenarios and produces deterministic results, eliminating the need for multi-run strategies.
2. **A 5-minute time budget is sufficient.** The solver plateaus within the first 1–2 minutes on all tested scenarios. Extending to 10 minutes provides negligible benefit. This aligns well with operational replanning windows.
3. **Adopt Orion as the primary automated planner.** Given its superior performance on the majority of scenarios and its deterministic output, Orion is the more practical choice for production deployment. The April 14 scenario showed that Baseline SA can occasionally find solutions Orion cannot reach, but this occurred only once in four attempts. Maintaining a single solver simplifies the development and operational workflow.
4. **Validate solutions against water level data before dispatch.** The current model assumes static vessel capacity. Until dynamic water level constraints are integrated, operators should cross-check Orion's output against current water level forecasts for draft-restricted segments, particularly along the Rhine corridor.

7.3. Future Research Directions

Several avenues for extending this work merit investigation:

Dynamic Water Level Constraints. The most impactful model enhancement would be incorporating real-time water level data to dynamically adjust vessel capacity along draft-restricted waterway segments. This would require a time-dependent capacity dimension that varies based on the projected water level at each bottleneck location at the vessel's estimated time of transit. Preliminary validation work, in which Orion's output was cross-referenced against historical water level data, found that only two routes

in the tested scenario produced water level violations. This suggests that, much like the Minimum Call Size constraint (which was ultimately disabled because the solver naturally satisfied it through travel cost minimization alone), water levels may be a constraint that the solver rarely violates in practice. If this pattern holds across more scenarios, a lightweight post-processing validation step may suffice as an interim solution, allowing a human planner to manually adjust the small number of conflicting routes rather than requiring a full time-dependent capacity model within the solver.

Improved Rolling-Horizon Replanning. Orion already supports a rolling-horizon workflow through its configurable start horizon and fixed-hours parameter, which locks stops within a specified window while allowing the solver to re-optimize the remaining schedule. However, further research could improve this mechanism, for example by warm-starting the local search from the previous solution rather than reconstructing from scratch, or by developing strategies that minimize disruption to already-communicated schedules when new orders arrive.

Fleet Reduction Strategies. The April 14 analysis revealed that neither the default atomic operators (relocate, swap, 2-opt) nor OR-Tools' built-in `FULL_PATH_LNS` operator, which destroys an entire route, could achieve fleet reduction under the problem's tight constraints. The generic route-destruction strategy lacks domain awareness: it does not know *which* vessel to target or how to redistribute orders feasibly. Investigating custom Large Neighbourhood Search operators that explicitly target fleet reduction, for example by identifying the least-utilized vessel and selectively redistributing its orders to compatible vessels with available capacity, could close this gap.

Data-Driven Enhancements. Machine learning could complement the solver in areas where OR-Tools' built-in heuristics have limited flexibility. For example, a learned model could predict promising initial vessel assignments based on historical planning data, providing the construction heuristic with a better starting point. Similarly, travel time predictions incorporating weather, water level forecasts, and historical delay patterns could replace the current fixed-speed assumption with more realistic estimates.

Hub-Based Consolidation. A growing trend in container shipping is the use of hub terminals to reduce congestion at major ports. Rather than having each vessel enter a congested port individually, orders destined for terminals within the port are first dropped at a designated hub terminal located in close proximity but outside the port. A dedicated vessel, loaded to maximum capacity, then consolidates these orders and enters the port in a single, efficient trip. This approach reduces total time spent inside the port (where turnaround times can exceed a full day) and benefits terminals, which prefer to handle fewer, fuller vessels. Extending Orion to support hub-based consolidation would require modelling transshipment points where orders can change vessels mid-route, as well as an analysis layer to evaluate which terminal yields the greatest cost savings when designated as a hub.

Alternative Order Aggregation Strategies. The current aggregation strategy groups orders that share identical origin, destination, time window, and container type. While this approach achieved a roughly 90% reduction in node count, it is not the only viable strategy. For example, Baseline SA operates at the individual order level and groups orders at the stop level, allowing orders with overlapping but non-identical time windows to be served at the same terminal visit. A relaxed aggregation strategy for Orion, one that groups orders with sufficiently overlapping time windows rather than requiring exact matches, could produce fewer but larger groups, further reducing the search space. However, such relaxation introduces a trade-off: the aggregated time window must be narrowed to the intersection of the individual windows, potentially over-constraining the solver. Investigating the sensitivity of solution quality to different aggregation strategies, and determining the degree of time window overlap required for effective grouping, is a promising direction for improving both scalability and solution quality.

A complementary direction concerns how orders are packed into chunks once they have been grouped by origin-destination pair. The current implementation uses a simple greedy strategy, but preliminary work has shown that replacing it with an exact bin-packing formulation using CP-SAT can produce higher-quality chunks. In this approach, orders sharing the same origin and destination are packed into a fixed number of bins subject to multi-dimensional capacity constraints (TEU, weight, reefer,

dangerous goods, and 45-foot containers), producing chunks that are guaranteed to be feasible for the smallest vessel in the fleet. Early results on one operator's dataset indicate that this capacity-aware packing improves upon the greedy baseline, though the magnitude of improvement has not yet been systematically quantified across all scenarios.

Synthetic Data and Stress Testing. The evaluation in this thesis was limited to six real-world scenarios. While these provide ecological validity and enable direct comparison with human plans, they do not systematically test the solver's behaviour under varying problem scales, constraint tightness, or fleet configurations. Generating synthetic problem instances with controlled parameters (e.g., number of orders, vessels, terminals, and time window widths) would enable stress testing and scalability analysis, revealing the boundaries of Orion's practical applicability. Generating such benchmarks would enable direct comparison with other solvers and methodologies in the VRP literature.

References

[1] Port of Rotterdam Authority, *Throughput figures*, Accessed: 2025, 2025. [Online]. Available: https://www.portofrotterdam.com/en/pressroom/throughput-figures.

[2] Central Commission for the Navigation of the Rhine, "Inland navigation in europe: Market observation annual report 2024," CCNR, Tech. Rep., 2024. [Online]. Available: https://inland-navigation-market.org/wp-content/uploads/2024/10/CCNR_annual_report_EN_2024_WEB.pdf.

[3] M. Buitendijk, *Antwerp Sets Minimum Call Size for Container Barges at Twenty | SWZ|Maritime — swzmaritime.nl*, https://swzmaritime.nl/news/2019/07/26/antwerp-sets-minimum-call-size-for-container-barges-at-twenty/, [Accessed 31-12-2025], 2019.

[4] P. Toth and D. Vigo, *The Vehicle Routing Problem*. Jan. 2002, ISBN: 9780898714982. DOI: 10.1137/1.9780898718515.

[5] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming — CP98*, M. Maher and J.-F. Puget, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 417–431, ISBN: 978-3-540-49481-2.

[6] F. Liu, C. Lu, L. Gui, Q. Zhang, X. Tong, and M. Yuan, *Heuristics for vehicle routing problem: A survey and recent advances*, 2023. arXiv: 2303.04147 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2303.04147.

[7] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2015.12.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835215004775.

[8] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, vol. 8, pp. 407–424, Dec. 2010. DOI: 10.1007/s10288-010-0130-z.

[9] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and A. Subramanian, "New benchmark instances for the capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017, ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2016.08.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221716306270.

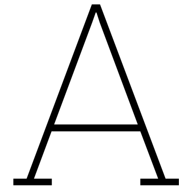
[10] J.-Y. Potvin and J.-M. Rousseau, "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows," *European Journal of Operational Research*, vol. 66, no. 3, pp. 331–340, 1993, ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(93)90221-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221793902218.

[11] T. Cuvelier, F. Didier, V. Furnon, S. Gay, S. Mohajeri, and L. Perron, "OR-Tools' Vehicle Routing Solver: a Generic Constraint-Programming Solver with Heuristic Search for Routing Problems," in *24e congrès annuel de la société française de recherche opérationnelle et d'aide à la décision*, ROADEF, Rennes, France, Feb. 2023. [Online]. Available: https://hal.science/hal-04015496.

[12] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Operations Research*, vol. 60, pp. 611–624, Jun. 2012. DOI: 10.1287/opre.1120.1048.

[13] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469–499, 1999, ISSN: 0377-2217. DOI: https://doi.org/10.1016/S0377-2217(98)00099-X. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S037722179800099X.

[14] A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 183–200, 2002. DOI: 10.1109/32.988498.



Core Implementation Listings

This appendix contains the specific Python implementation details for the constraints discussed in Chapter 5.

Listing A.1: Dynamic Time Callback Implementation

```
def time_callback(from_index, to_index, inv_speed=v_inv_speed):
    from_node = index_map[from_index]
    to_node = index_map[to_index]

    # 1. Travel Time (Distance * 1/Speed)
    dist = dist_matrix[from_node][to_node]
    travel_time = dist * inv_speed

    # 2. Handling Time (at 'from_node')
    h_time = handling_times[from_node]

    # 3. Berth Time (transition overhead if new terminal)
    bs_time = base_stop_times[from_node][to_node]

    return int(travel_time + h_time + bs_time + 0.5)
```

Listing A.2: Capacity Dimension Construction

```
# Register Unary Callback (returns demand at node)
evaluator_index = self.routing.RegisterUnaryTransitCallback(callback)

self.routing.AddDimensionWithVehicleCapacity(
    evaluator_index,
    0,  # null capacity slack (load must be exact)
    capacities, # Vector of max capacities per vessel
    False, # fix_start_cumul_to_zero (False for pre-loaded vessels)
    dimension_name,
)
```

Listing A.3: Charter Cost Evaluation Logic

```
# We evaluated multiple strategies to penalize vessel usage.
# STRATEGY 2: Explicit Activation Cost (Evaluated but Rejected)
# Penalize usage with the full cost of the planning horizon
total_fixed_cost = int(max_day_cost * horizon_days)
self.routing.SetFixedCostOfVehicle(total_fixed_cost, v_idx)

# STRATEGY 3: Implicit / Distance Minimization (Selected)
# We apply no specific charter penalty, allowing the solver to
# optimize purely for distance. This resulted in the lowest
# total operating cost during benchmarking.
pass
```

Listing A.4: Implementing the trucking cost trade-off

```
# Penalty represents the alternative cost (e.g., Trucking)
# If penalty is infinite (hard constraint), the order is mandatory.
is_optional = (penalty < 1_000_000_000_000)

if is_optional:
    # Add Disjunction: Solver chooses between "Visit" (Cost) or "Skip" (Penalty)
    self.routing.AddDisjunction(p_routing_indices, int(penalty))
    # Delivery is linked to Pickup; if Pickup is skipped, Delivery is skipped.
    self.routing.AddDisjunction(d_routing_indices, 0)
```

Listing A.5: Binding delivery nodes to specific vessels

```
# Iterate through all onboard delivery nodes
for order_idx, node_indices in self.data.nodes.onboard_delivery_nodes.items():
    # Identify the vessel currently carrying the cargo
    first_node = self.data.nodes.locations[node_indices[0]]
    bound_vessel_idx = first_node.vessel_idx

    if bound_vessel_idx is None:
        continue

    # Convert graph node indices to routing solver indices
    routing_indices = [self.manager.NodeToIndex(i) for i in node_indices]

    # Force the solver to assign this node ONLY to the bound vessel
    # This prunes the search space by removing all other vehicles from the domain
    allowed_vehicles = [bound_vessel_idx]
    for idx in routing_indices:
        self.routing.SetAllowedVehiclesForIndex(allowed_vehicles, idx)
```

Listing A.6: MCS Linear Penalty Calculation

```
# Calculate Deficit: max(0, MCS - Volume)
# is_active is derived from: volume > 0
deficit_raw = (is_active * int(mcs)) - total_volume_v

# Ensure deficit is non-negative
deficit_var = solver.IntVar(
    0, int(mcs), f"mcs_deficit_{terminal_id}_{v_idx}"
)
solver.Add(deficit_var >= deficit_raw)

# Apply penalty proportional to the deficit
penalty_term = deficit_var * unit_cost
self.mcs_penalties[v_idx].append(penalty_term)
```

Listing A.7: Effective Overtime Detection Logic

```
# EffectiveOvertime = InFlex AND (NOT InRegular)
# Using arithmetic multiplication for boolean AND
is_time_overtime = is_in_flex * (solver.IntConst(1) - is_in_regular)

# Node is Overtime if VisitedByV AND TimeIndex is EffectiveOvertime
is_node_in_overtime = solver.Min(is_visited_by_v, is_time_overtime)

# Calculate Overtime Volume Contribution
# Only count volume if it is handled IN OVERTIME
overtime_volume_terms.append(is_node_in_overtime * abs_teu)
```

Listing A.8: Enforcing Break Intervals

```
# Create Fixed Interval for the break
fixed_break = solver.FixedInterval(
    int(s_min), int(duration_min), f"Break_V{v_idx}_{s_min}"
)
break_intervals.append(fixed_break)

# Apply breaks to Time dimension
# node_visit_transits ensures breaks don't interrupt handling
```

```
time_dim.SetBreakIntervalsOfVehicle(
    break_intervals, v_idx, node_visit_transits
)
```

Listing A.9: Forbidden Terminal Domain Reduction

```
if term_id in forbidden_ids:
    # Vehicle v_idx strictly cannot visit node i
    index = self.manager.NodeToIndex(i)
    self.routing.VehicleVar(index).RemoveValue(v_idx)
```

Listing A.10: Strict Domain Reduction Strategy

```
# Identify gaps between valid operational windows
# e.g., closing at 23:00 on Day 1 and opening at 06:00 on Day 2
for k in range(len(valid_windows) - 1):
    gap_start = valid_windows[k].end + 1
    gap_end = valid_windows[k+1].start - 1

    # Strictly remove the closed interval from the domain
    # This prevents the solver from scheduling visits during overnight breaks
    cumul_j.RemoveInterval(int(gap_start), int(gap_end))
```

B

Automated Infeasibility Diagnosis

As discussed in Chapter 5, the complexity of the VRPPDTW constraints, specifically the interaction between time windows, sailing distances, and mandatory breaks, can lead to scenarios where the solver returns a status of *Infeasible*. In large datasets involving hundreds of orders, identifying the specific subset of conflicting requirements is a non-trivial task. To automate this diagnosis, we implemented a custom debugging tool based on the Delta Debugging algorithm [14].

B.1. Theoretical Background

The debugging tool is grounded in the *Minimizing Delta Debugging* ($ddmin$) algorithm introduced by Zeller and Hildebrandt [14]. The core premise is to simplify a failing test case c_χ (where constraints cannot be met) into a 1-minimal test case, such that removing any single input entity would cause the failure to disappear (i.e., the problem becomes feasible).

Formally, let c_χ be the set of input configurations (orders) that produces a failure. The algorithm partitions c_χ into subsets $\Delta_1, \dots, \Delta_n$. It then iteratively tests these subsets and their complements ($\nabla_i = c_\chi - \Delta_i$) to reduce the search space. The algorithm operates on three outcomes:

- **Reduce to Subset:** If a subset Δ_i fails (is infeasible), the search continues within Δ_i with $n = 2$.
- **Reduce to Complement:** If the complement ∇_i fails, the conflict lies outside Δ_i. The search continues within ∇_i with $n - 1$.
- **Increase Granularity:** If no subset or complement fails explicitly (often due to interference between disjoint elements), the granularity n is doubled to test smaller chunks.

B.2. Implementation Strategy

We adapted the generic $ddmin$ algorithm to the specific domain of VRPPDTW. The implementation, written in Python, interacts directly with the `Solver` and `Modeler` classes defined in Chapter 5.

B.2.1. Mapping Inputs to the Zeller Model

The abstract components of the Zeller algorithm were mapped to the solver architecture as follows:

- **Circumstances (c):** The set of circumstances is defined as the list of active `orderId` strings provided to the solver.
- **Test Function ($test(c)$):** The function `clean_and_solve` acts as the test oracle. It constructs a temporary `Scenario` object containing only the subset of orders and attempts to solve it with a short time limit (5 seconds).
- **Outcomes:**

- χ (Fail): The solver returns `None` or status `INFEASIBLE`. This indicates the subset retains the "toxic" conflicting logic.
- $\surd$ (Pass): The solver returns a valid solution. This indicates the subset is feasible.

B.2.2. Process Isolation

A critical practical consideration was the stability of the Constraint Programming solver during rapid iterative testing. To prevent memory leaks or internal state corruption in the C++ backend of OR-Tools, each test run is executed in an isolated process using Python's `multiprocessing` library.

Listing B.1: Process isolation in the test function

```
def solve_worker(data_subset, df_terminals, df_distances, time_limit=5):
    try:
        scenario = Scenario(**data_subset)
        modeler = Modeler(scenario, df_terminals, df_distances)
        # ... solver instantiation ...
        _, solution = solver.solve()
        return True if solution else False
    except Exception:
        return False
```

B.2.3. Algorithmic Logic

The core logic resides in the `run_ddmin` method. It implements the "Divide and Conquer" approach described by Zeller. It utilizes a helper method `split_list` to handle the partitioning of order IDs into n chunks.

Listing B.2: The minimizing delta debugging loop

```
while len(candidates) >= 1:
    chunks = self.split_list(candidates, n)
    some_subset_failed = False

    # 1. Test Chunks (Reduce to Subset)
    for i, chunk in enumerate(chunks):
        if not self.clean_and_solve(chunk, f"Chunk {i}"):
            candidates = chunk # Found smaller toxic set
            n = 2
            some_subset_failed = True
            break

    # 2. Test Complements (Reduce to Complement)
    if not some_subset_failed and n > 1:
        for i, chunk in enumerate(chunks):
            complement = [x for x in candidates if x not in chunk]
            if not self.clean_and_solve(complement):
                candidates = complement # Conflict is in the remainder
                n = max(n - 1, 2)
                some_subset_failed = True
                break

    # 3. Increase Granularity
    if not some_subset_failed:
        if n < len(candidates):
            n = min(len(candidates), n * 2)
        else:
            break
```

B.3. Output: The Toxic Scenario

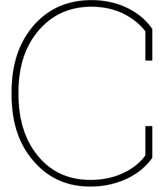
Upon convergence, the tool identifies a minimal set of orders that are provably infeasible. The tool automatically exports this subset as a standalone JSON file, `toxic_scenario.json`.

This artifact allows the developer to inspect the specific constraints of the conflicting orders (e.g., an order with a pickup at 10:00 requiring a 4-hour transit to a delivery closing at 13:00) without the noise of the hundreds of feasible orders surrounding them.

B.4. AI-Assisted Remediation

Since the Delta Debugging tool isolates the "toxic" scenario into a small, self-contained JSON file, it enables a powerful debugging workflow using AI agents (e.g., Google Antigravity).

Instead of manually tracing the solver's internal logs, the developer can provide the minimal `toxic_scenario.json` and the relevant constraint code to an AI agent. Because the input is minimal (often 1-2 orders), the agent can effectively "play" with the constraints, adjusting time windows, disabling specific logic blocks, or simulating the routing variables, to identify strictly logical contradictions or implementation bugs. This feedback loop significantly accelerates the resolution of feasibility issues compared to traditional manual debugging of the full dataset.



Detailed Experimental Results

This appendix contains the full experimental data for both phases of the evaluation.

C.1. Phase 1 Results

Phase 1 consists of 120 runs (10 configurations × 6 scenarios × 2 runs each), each lasting 5 minutes.

Table C.1: Detailed results for all experimental runs.

Scenario	Config	Run	Sols	Stops	Init Obj	Final Obj	0-30s	30-60s	Min2	Min3	Min4	Min5	Unassigned	Travel Cost
August 4	LCI + TS	1	569	33	2,045,614	107,926	99.96%	0.84%	0.00%	0.00%	0.00%	0.00%	0	104,826
August 4	LCI + TS	2	569	33	2,045,614	107,926	99.96%	0.84%	0.00%	0.00%	0.00%	0.00%	0	104,826
August 4	LCI + GLS	1	1524	32	2,045,614	115,716	99.95%	0.19%	3.68%	0.00%	0.00%	0.00%	0	112,716
August 4	LCI + GLS	2	1518	32	2,045,614	115,716	99.95%	0.19%	3.68%	0.00%	0.00%	0.00%	0	112,716
August 4	PCI + TS	1	522	34	9,150,265	115,916	99.95%	9.53%	0.62%	0.23%	0.00%	0.00%	0	112,716
August 4	PCI + TS	2	516	34	9,150,265	115,916	99.95%	9.53%	0.62%	0.23%	0.00%	0.00%	0	112,716
August 4	PCI + GLS	1	1196	37	9,150,265	116,898	99.95%	13.32%	0.84%	0.19%	0.00%	0.00%	0	113,298
August 4	PCI + GLS	2	1259	37	9,150,265	116,898	99.95%	0.74%	0.92%	0.11%	0.00%	0.00%	0	113,298
August 4	LCI + GTS	1	28	38	2,045,614	122,130	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	118,830
August 4	LCI + GTS	2	28	38	2,045,614	122,130	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	118,830
August 4	LCI + GD	1	14	43	2,045,614	126,617	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	122,917
August 4	LCI + GD	2	14	43	2,045,614	126,617	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	122,917

Continued on next page

Table C.1 – continued from previous page

Scenario	Config	Run	Sols	Stops	Init Obj	Final Obj	0-30s	30-60s	Min2	Min3	Min4	Min5	Unassigned	Travel Cost
August 4	LCI + SA	1	547	43	2,045,614	126,617	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	122,917
August 4	LCI + SA	2	556	43	2,045,614	126,617	99.95%	0.00%	0.00%	0.00%	0.00%	0.00%	0	122,917
August 4	PCI + GD	1	7	43	9,150,265	8,938,082	96.43%	0.00%	0.00%	0.00%	0.00%	0.00%	191	133,882
August 4	PCI + GD	2	7	43	9,150,265	8,938,082	96.43%	0.00%	0.00%	0.00%	0.00%	0.00%	191	133,882
August 4	PCI + SA	1	443	43	9,150,265	8,938,082	96.43%	0.00%	0.00%	0.00%	0.00%	0.00%	191	133,882
August 4	PCI + SA	2	441	43	9,150,265	8,938,082	96.43%	0.00%	0.00%	0.00%	0.00%	0.00%	191	133,882
August 4	PCI + GTS	1	23	43	9,150,265	138,486	99.94%	0.00%	0.00%	0.00%	0.00%	0.00%	0	134,286
August 4	PCI + GTS	2	23	43	9,150,265	138,486	99.94%	0.00%	0.00%	0.00%	0.00%	0.00%	0	134,286
April 14	LCI + TS	1	389	37	5,152,685	107,505	99.96%	10.23%	0.00%	0.00%	0.70%	0.00%	0	104,205
April 14	LCI + TS	2	397	37	5,152,685	107,505	99.96%	10.23%	0.00%	0.00%	0.70%	0.00%	0	104,205
April 14	LCI + GLS	1	1038	37	5,152,685	112,241	99.96%	2.12%	0.00%	4.92%	0.09%	0.00%	0	108,841
April 14	LCI + GLS	2	1045	37	5,152,685	112,241	99.96%	2.12%	0.00%	4.92%	0.09%	0.00%	0	108,841
April 14	PCI + TS	1	321	38	21,032,527	115,052	99.95%	0.00%	0.51%	5.18%	0.08%	3.63%	0	111,553
April 14	PCI + TS	2	323	38	21,032,527	115,052	99.95%	0.00%	0.51%	5.18%	0.08%	3.63%	0	111,553
April 14	LCI + GTS	1	28	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	LCI + GTS	2	28	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	LCI + GD	1	20	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	LCI + GD	2	20	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	LCI + SA	1	795	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	LCI + SA	2	787	42	5,152,685	120,713	99.96%	0.00%	0.00%	0.00%	0.00%	0.00%	0	117,013
April 14	PCI + GLS	1	1044	38	21,032,527	126,413	98.78%	0.00%	96.31%	0.00%	0.00%	0.00%	0	122,913
April 14	PCI + GLS	2	1042	38	21,032,527	126,413	98.78%	0.00%	96.31%	0.00%	0.00%	0.00%	0	122,913
April 14	PCI + GTS	1	16	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
April 14	PCI + GTS	2	16	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
April 14	PCI + GD	1	10	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
April 14	PCI + GD	2	10	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
April 14	PCI + SA	1	371	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
April 14	PCI + SA	2	376	38	21,032,527	8,730,383	96.89%	0.00%	0.00%	0.00%	0.00%	0.00%	86	126,883
March 23	LCI + GLS	2	166	102	51,219	31,785	99.94%	12.88%	6.30%	0.00%	0.00%	0.00%	0	23,085
March 23	PCI + TS	1	177	101	50,375	32,334	99.94%	5.44%	1.18%	4.16%	0.00%	0.00%	0	23,534
March 23	PCI + TS	2	178	100	50,375	32,334	99.94%	5.32%	1.31%	4.16%	0.00%	0.00%	0	23,534
March 23	LCI + SA	2	207	97	53,621	32,506	99.94%	16.52%	0.00%	0.00%	0.00%	0.00%	0	23,806
March 23	LCI + TS	1	100	97	51,219	32,842	99.94%	13.89%	0.00%	0.00%	0.00%	0.00%	0	24,142
March 23	LCI + GLS	1	192	97	51,219	32,985	99.93%	16.52%	0.00%	0.00%	0.00%	0.00%	0	24,185
March 23	LCI + GD	2	95	96	53,621	33,081	99.93%	16.28%	0.00%	0.00%	0.00%	0.00%	0	24,281
March 23	LCI + GTS	1	110	99	52,037	33,432	99.94%	10.64%	0.00%	0.00%	0.00%	0.00%	0	24,632
March 23	PCI + GLS	1	203	104	50,375	33,945	99.94%	5.32%	0.23%	0.00%	0.48%	0.00%	0	24,945
March 23	PCI + GLS	2	203	104	50,941	33,945	99.94%	5.32%	0.23%	0.00%	0.48%	0.00%	0	24,945

Continued on next page

Table C.1 – continued from previous page

Scenario	Config	Run	Sols	Stops	Init Obj	Final Obj	0-30s	30-60s	Min2	Min3	Min4	Min5	Unassigned	Travel Cost
March 23	PCI + GTS	2	131	104	50,375	34,108	99.94%	5.32%	0.23%	0.00%	0.00%	0.00%	0	25,108
March 23	PCI + GD	2	129	104	50,375	34,108	99.94%	9.15%	0.23%	0.00%	0.00%	0.00%	0	25,108
March 23	PCI + SA	1	227	105	50,375	34,108	99.94%	5.50%	0.09%	0.00%	0.00%	0.00%	0	25,108
March 23	LCI + SA	1	256	98	53,621	35,899	99.94%	7.02%	0.84%	0.00%	0.00%	0.00%	0	26,999
March 23	PCI + GD	1	126	103	50,375	36,048	99.94%	6.74%	0.00%	0.00%	0.00%	0.00%	0	27,048
March 23	LCI + TS	2	128	100	51,219	36,217	99.94%	6.98%	0.00%	0.00%	0.00%	0.00%	0	27,217
March 23	LCI + GD	1	75	100	51,219	36,399	99.94%	6.52%	0.00%	0.00%	0.00%	0.00%	0	27,399
March 23	PCI + GTS	1	119	106	50,375	37,167	99.93%	4.79%	0.00%	0.00%	0.00%	0.00%	0	28,067
March 23	PCI + SA	2	227	105	50,375	37,167	99.94%	4.71%	0.00%	0.00%	0.00%	0.00%	0	28,067
March 23	LCI + GTS	2	107	102	57,135	39,875	99.92%	21.35%	0.00%	0.00%	0.00%	0.00%	0	30,875
October 27	LCI + TS	1	77	67	35,335	20,812	99.92%	26.65%	0.00%	6.44%	0.00%	0.00%	0	15,213
October 27	LCI + TS	2	74	66	35,335	20,812	99.92%	26.47%	0.00%	6.44%	0.00%	0.00%	0	15,213
October 27	LCI + GTS	1	46	66	35,335	22,245	99.92%	26.65%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + GTS	2	46	66	35,335	22,245	99.92%	26.65%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + GD	1	44	66	35,335	22,245	99.92%	26.83%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + GD	2	44	66	35,335	22,245	99.92%	26.65%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + GLS	1	113	66	35,335	22,245	99.92%	26.47%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + GLS	2	106	66	35,335	22,245	99.91%	21.66%	9.46%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + SA	1	121	65	35,335	22,245	99.91%	10.99%	20.30%	0.00%	0.00%	0.00%	0	16,645
October 27	LCI + SA	2	119	66	35,335	22,245	99.92%	26.65%	0.00%	0.00%	0.00%	0.00%	0	16,645
October 27	PCI + SA	1	133	63	34,635	26,427	99.92%	6.73%	3.51%	0.00%	0.00%	0.00%	0	20,627
October 27	PCI + SA	2	133	63	34,635	26,427	99.92%	6.73%	3.51%	0.00%	0.00%	0.00%	0	20,627
October 27	PCI + GLS	1	89	68	34,635	28,954	99.92%	0.66%	0.00%	0.00%	0.00%	0.74%	0	22,754
October 27	PCI + GLS	2	88	68	34,635	28,954	99.92%	0.66%	0.00%	0.00%	0.00%	0.74%	0	22,754
October 27	PCI + GTS	1	30	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
October 27	PCI + GTS	2	30	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
October 27	PCI + GD	1	28	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
October 27	PCI + GD	2	28	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
October 27	PCI + TS	1	70	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
October 27	PCI + TS	2	70	69	34,635	29,170	99.92%	0.66%	0.00%	0.00%	0.00%	0.00%	0	22,870
April 28	LCI + GTS	1	217	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + GTS	2	222	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + GD	1	4	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + GD	2	4	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + GLS	1	20092	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + GLS	2	20193	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + SA	1	11109	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + SA	2	11108	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630

Continued on next page

Table C.1 – continued from previous page

Scenario	Config	Run	Sols	Stops	Init Obj	Final Obj	0-30s	30-60s	Min2	Min3	Min4	Min5	Unassigned	Travel Cost
April 28	LCI + TS	1	15244	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	LCI + TS	2	15228	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GTS	1	217	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GTS	2	209	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GD	1	1	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GD	2	1	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GLS	1	18490	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + GLS	2	17192	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + SA	1	31421	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + SA	2	30593	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + TS	1	14209	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
April 28	PCI + TS	2	14884	7	133,325	133,325	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0	132,630
March 30	LCI + TS	1	255	68	37,285	24,317	99.97%	0.00%	2.66%	0.00%	0.00%	0.00%	0	17,618
March 30	LCI + GTS	1	85	71	37,285	24,980	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	18,180
March 30	LCI + GD	1	77	71	36,764	25,373	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	18,673
March 30	LCI + GD	2	71	75	38,461	26,185	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	19,085
March 30	LCI + GLS	1	376	75	36,194	26,102	99.97%	0.00%	0.00%	0.97%	0.00%	0.00%	0	19,102
March 30	LCI + TS	2	251	77	35,315	26,711	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	19,411
March 30	PCI + TS	1	257	76	32,281	26,722	99.97%	0.00%	0.00%	3.71%	0.00%	3.76%	0	19,423
March 30	LCI + GLS	2	421	76	35,521	27,425	99.97%	0.41%	0.00%	0.00%	0.00%	0.00%	0	20,325
March 30	LCI + GTS	2	55	78	36,155	27,862	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	20,562
March 30	LCI + SA	2	237	78	36,174	28,274	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	20,674
March 30	PCI + TS	2	246	78	32,281	27,987	99.97%	0.00%	0.00%	0.00%	2.16%	0.41%	0	20,887
March 30	LCI + SA	1	279	73	37,309	28,578	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,478
March 30	PCI + GTS	1	36	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + GTS	2	36	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + GD	1	31	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + GD	2	32	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + GLS	1	373	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + GLS	2	373	78	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + SA	1	241	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522
March 30	PCI + SA	2	240	77	32,281	28,722	99.97%	0.00%	0.00%	0.00%	0.00%	0.00%	0	21,522

C.2. Phase 2 Results

Phase 2 features the best 2 configurations (LCI + TS and LCI + GLS) run for 10 minutes on each scenario once.

Table C.2: Detailed results for Phase 2 (10-minute runs).

Scenario	Config	Sols	Stops	Init Obj	Final Obj	0-30s	30-60s	Min2	Min3	Min4	Min5	Min6	Min7	Min8	Min9	Min10	Travel Cost
August 4	LCI + TS	1043	33	2,045,614	107,926	99.96%	0.84%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	104,826
August 4	LCI + GLS	2705	33	2,045,614	115,592	99.95%	0.19%	3.68%	0.00%	0.00%	0.00%	0.02%	0.00%	0.00%	0.09%	0.00%	112,692
April 14	LCI + TS	779	37	5,152,685	107,505	99.96%	10.23%	0.00%	0.00%	0.70%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	104,205
April 14	LCI + GLS	2037	40	5,152,685	111,672	99.96%	2.12%	0.00%	4.92%	0.09%	0.00%	0.00%	0.46%	0.00%	0.00%	0.05%	108,072
March 23	LCI + TS	187	100	54,807	31,978	99.92%	17.68%	0.00%	18.19%	0.84%	0.00%	0.00%	0.00%	0.00%	0.00%	0.63%	23,378
March 23	LCI + GLS	232	97	53,621	33,081	99.93%	17.75%	2.03%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	24,281
October 27	LCI + TS	120	66	35,335	20,812	99.92%	26.47%	0.00%	6.44%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	15,213
October 27	LCI + GLS	178	65	35,335	22,245	99.92%	26.47%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	16,645
April 28	LCI + GLS	36352	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	132,630
April 28	LCI + TS	27534	7	197,621	133,325	32.54%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	132,630
March 30	LCI + TS	289	74	36,174	25,686	99.97%	4.27%	3.55%	0.00%	0.00%	0.00%	0.00%	0.00%	1.23%	0.00%	0.00%	18,586
March 30	LCI + GLS	584	72	36,174	26,356	99.97%	0.49%	0.00%	0.00%	0.00%	0.00%	0.00%	0.59%	0.00%	0.00%	0.00%	19,256

C.3. Extended Baseline SA Results

To complement the original four 5-minute runs, Baseline SA was run twice more per scenario for extended durations. Because Baseline SA uses iteration-based termination rather than a wall-clock time limit, runtimes vary between 2 minutes 37 seconds and 8 minutes 52 seconds. The convergence percentages below represent the period-over-period reduction in Baseline SA's internal objective, which includes large penalty terms for constraint violations. These values are therefore not directly comparable to Orion's convergence columns.

Table C.3: Detailed results for extended Baseline SA runs.

Scenario	Run	Runtime	0-30s	30-60s	Min2	Min3	Min4	Min5	Min6	Min7	Min8	Min9	Travel Cost
August 4	1	8m 24s	31.14%	94.02%	76.57%	8.83%	0.01%	0.00%	0.00%	0.00%	0.00%	—	132,363
August 4	2	8m 52s	21.94%	38.69%	97.32%	95.27%	99.85%	8.79%	0.00%	0.00%	0.00%	—	129,092
April 14	1	5m 53s	38.68%	74.58%	99.99%	5.19%	19.21%	3.75%	3.72%	—	—	—	88,813
April 14	2	7m 03s	43.58%	99.95%	99.60%	12.15%	1.45%	5.41%	8.35%	0.05%	—	—	87,583
March 23	1	8m 16s	57.38%	96.62%	33.67%	6.05%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	28,332
March 23	2	8m 17s	49.98%	97.06%	2.70%	6.48%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	26,326
October 27	1	8m 43s	26.44%	97.31%	95.10%	99.99%	4.63%	0.00%	6.97%	13.85%	4.30%	3.70%	16,162
October 27	2	7m 41s	10.11%	71.91%	99.63%	95.22%	99.74%	0.00%	11.10%	1.28%	0.84%	—	19,520
April 28	1	2m 37s	0.71%	0.00%	0.00%	0.00%	—	—	—	—	—	—	169,626
April 28	2	5m 53s	22.37%	0.00%	0.14%	0.00%	0.00%	0.00%	0.00%	—	—	—	132,627
March 30	1	4m 57s	73.14%	16.51%	0.77%	0.00%	0.00%	0.00%	—	—	—	—	24,007
March 30	2	4m 42s	71.77%	29.68%	0.80%	0.03%	0.00%	0.00%	—	—	—	—	22,974