# Optimisation of Safety Stock and Reorder Points for Multi-item Multilocation Inventory Networks

Lou Ann Abellaneda

Bachelor's thesis
2026:K20

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

# Abstract

Spare parts inventory management in multi-echelon supply chains presents a significant optimisation challenge, particularly in the commercial vehicle aftermarket where part availability directly affects vehicle uptime and customer retention.

This thesis develops and evaluates a Mixed-Integer Linear Programming (MILP) model for jointly optimising reorder points and order quantities across a three-echelon network consisting of a Central Distribution Centre (CDC), a Support Distribution Centre (SDC), and Dealers, for multiple items over a discrete planning horizon.

The model is formulated under two objective functions: minimisation of total backorder volume across all locations and periods, and minimisation of total supply chain cost comprising of backorder penalty costs, inventory holding costs, order-line costs, and cost of lost sales at dealer level. Both formulations are subject to operational constraints including rounding requirements, full-truckload volumetric capacity, order-line thresholds, and a historical inventory value cap. The model is implemented using the Gurobi Optimizer [7] and solved on an academic problem instance involving two items, three locations, and five planning periods.

Limitations including deterministic demand and a single replenishment channel per location are acknowledged and directions for future work are identified.

# Popular Science Summary

Imagine a truck breaking down on a highway carrying a full load of goods. Every hour it sits idle, the operator loses substantial amounts of money. The part needed to fix it might be sitting in a warehouse on the other side of the country, or it might have run out last week. The difference between those two outcomes comes down to one question: how much of each spare part should you keep, and where?

Commercial vehicle manufacturers like Volvo face this question at enormous scale. They manage thousands of different parts, from simple filters to complex engine components, spread across a network of central warehouses, regional hubs, and local dealers across the world. Stock too little of a critical part and a customer's truck stays broken; stock too much and capital is tied up in inventory that may sit on shelves for months. Getting this balance right is not just an operational concern — it directly affects whether a fleet operator chooses to stay with a supplier or switch to a competitor.

This thesis asks whether modern mathematical optimisation can do better than the formulas currently used to set inventory levels. The approach, called Mixed-Integer Linear Programming (MILP), works by translating the entire inventory problem (what to order, how much, when, and where to hold it) into a mathematical model that a computer can solve to find the best possible answer. The model covers a three-level supply chain: a central distribution centre, a support distribution centre, and a network of dealers, all connected by different lead times and transport costs. It accounts for real-world constraints like the fact that parts must be ordered in specific quantities, trucks have a maximum load, and each warehouse has a budget ceiling on how much inventory it can hold.

Two different goals were tested: minimising the number of backorders (when a part is needed but not available) and minimising total cost (balancing the expense of holding stock, placing orders, and paying penalties when parts run out). On a simplified test case, the model solved to near-optimal quality in under a tenth of a second, fast enough to be useful in practice.

When applied to real data from Volvo, the model recommended much lower reorder points and order quantities than Volvo currently uses — reductions of up to 94% in some cases. This reveals something important: the model was given perfect information. It knew exactly how much demand to expect and was told that suppliers always deliver on time. In the real world, neither of those things is true. Volvo's higher stock levels reflect the uncertainty the model was allowed to ignore: the risk of a delayed shipment, a demand spike, or a supplier falling short. The gap between the model's recommendations and Volvo's actual settings is, in a sense, the price of uncertainty.

This points clearly to the next step: building uncertainty into the model. If the computer is also told that demand fluctuates and deliveries sometimes arrive late, it will recommend safety stock to buffer against those risks, bringing its suggestions closer to what real-world operations require. The mathematical tools for doing this already exist, and the groundwork laid in this thesis (the model structure, the solver setup, and the sensitivity tests) provides the foundation on which that more realistic model can be built.

The broader message is that inventory management, which can seem like a matter of

intuition and experience, is also a mathematical problem with better and worse answers. Getting closer to the best answer, even by a few percentage points in service levels or capital efficiency, translates directly into trucks staying on the road and customers staying satisfied.

# Acknowledgements

I would like to thank my supervisors Iulian Carpatorea at Volvo Group and Dr. Mengwu Guo at Lund University for their help and guidance throughout my thesis. Thank you Marcus Bohman for giving me the opportunity to write my thesis at Volvo. And finally thank you Philip Mårtensson, Rasmus Asklund and Carl Munck for consolidating my knowledge and helping me understanding more clearly the supply chain environment at Volvo Group.

Large Language Models (Cursor, Claude) were used to help generate LaTeX formatting commands and used as support for the coding.

# Contents

# 1. Introduction

## 1.1 Background and Industrial Motivation

Spare parts inventory management represents a critical operational challenge for Volvo, driven by four interdependent pressures. First, vehicle downtime carries severe financial consequences: a single commercial truck off the road can generate losses of hundreds to thousands of euros per hour through halted logistics, missed deliveries, and contractual penalties, meaning part availability directly translates to customer value. Second, the sheer diversity of stock keeping units, spanning standard consumables to highly specific engine components, each with distinct demand patterns, lead times, and criticality levels, makes manual or heuristic-based planning systematically unreliable at scale. Third, the working capital tied up in spare part inventories across a multi-echelon network is substantial, and inefficient stocking decisions have direct balance sheet implications. Fourth, in the commercial vehicle aftermarket, service level is a genuine competitive differentiator: fleet operators select suppliers based in part on uptime guarantees, meaning fill rate performance shapes long-term customer retention and brand positioning.

Together, these pressures make inventory optimisation not merely an operational concern but a strategic, which motivates a rigorous, systematic approach rather than one based on managerial intuition or simplified rules of thumb.

## 1.2 Current Practice at Volvo Group

At the supply chain division at Volvo Group, the current inventory and safety stock practice is based on a single-echelon continuous-review $(R, Q)$ replenishment model, where replenishment is triggered when inventory position reaches a reorder point. The system combines Excel for planning with a Python optimization engine to calculate optimal safety stock and reorder points using both cost and service-level considerations.

The practice differs by dealer type. Some dealers use a formula-driven approach based mainly on demand variability, lead time, and "picks" (fulfilled order lines) as the demand signal, while other dealers use target service levels (e.g., 95%) and back-calculate the required safety stock stochastically. Different parts use different distributions based on their characteristics to better capture their demand characteristics.

Optimization is performed by balancing holding, ordering, and shortage costs while tracking both cycle service level and fill rate. The current setup also includes periodic reoptimization triggers when demand patterns, service targets, or performance metrics change. The current implementation supports both single-location and multi-echelon inventory optimization (MEIO), providing Volvo with the flexibility to optimize inventory independently at each location or holistically across the supply chain to account for upstream stockouts and induced backorders

## 1.3 Literature Review

### 1.3.1 Inventory Theory and Safety Stock

The classical $(R, Q)$ reorder point policy under deterministic demand traces its origins to the Economic Order Quantity model of Harris (1913) and Wilson (1934), as described in [1]. While foundational, these single-echelon, deterministic formulations are ill-suited to the multi-stage, stochastic environments characteristic of modern supply chains. The decisive theoretical advance came with Clark and Scarf (1960), who proved that echelon-stock base policies are optimal for serial systems and demonstrated that optimal base-stock levels can be recovered by decomposing the problem into a sequence of single-echelon subproblems. This decomposition result remains the theoretical backbone of most subsequent multi-echelon work and directly motivates the three-echelon formulation adopted in this thesis, though it is worth noting that its optimality guarantees rely on assumptions, such as linear holding and penalty costs and uncapacitated stages, that do not always hold in practice.

The literature has since merged around two dominant modeling paradigms, as described by [5]: the stochastic-service model (SSM), originating with Clark and Scarf (1960), and the guaranteed-service model (GSM), introduced by Simpson (1958). These approaches differ fundamentally in how they treat demand propagation, material flow, and service time characteristics across echelons. The SSM explicitly models stockout risk and backorder propagation, making it more analytically faithful to real system behaviour but computationally demanding; the GSM instead assumes worst-case demand is bounded and service times are committed in advance, yielding tractable optimisation problems at the cost of potentially conservative safety stock estimates. The choice between these paradigms is not merely technical: it carries direct implications for the tightness of cost bounds and the realism of service level guarantees. This thesis adopts the SSM framing, since the automotive aftermarket setting involves genuinely stochastic and intermittent demand that cannot be credibly bounded.

The most comprehensive survey of this field is a systematic review of 95 articles from 1977 to 2019 by [6], the first to focus specifically on Operations Research-based approaches to safety stock dimensioning. It documents the field's trajectory across modelling approaches, industrial applications, solution techniques, and performance criteria. A recurring finding is that single-echelon approaches, which treat each stage independently and ignore the interdependence of service level and cost performance across customer-supplier boundaries, tend to produce either poor customer service or non-cost-optimal solutions [5]. Multi-echelon approaches, by optimising inventory decisions across all stages simultaneously, consistently outperform their single-echelon counterparts on both dimensions, though the gains come at computational cost that scales with problem size and structural complexity. This cost-performance tradeoff is a central concern of the present work.

### 1.3.2 Mathematical Programming in Inventory Optimisation

Analytical methods such as base-stock level formulae, while elegant, break down when real-world constraints are introduced: capacity limits, service commitments, and mixed replenishment structures resist closed-form treatment. Simulation-based and heuristic methods offer flexibility but sacrifice optimality guarantees, making it difficult to benchmark solutions or certify performance. Mathematical programming fills this gap by

obtaining provably optimal solutions within explicitly stated model boundaries, albeit at a computational cost that grows with problem scale, a tradeoff investigated directly in [11].

MILP in particular has become the tool of choice for inventory problems with integer structure [8], enabling precise modelling of operational constraints including production capacities, inventory limits, transportation schedules, and demand fulfilment requirements. The framework accommodates the kind of combinatorial structure that arises naturally in multi-echelon spare parts settings, and commercial solvers such as Gurobi Optimizer [7], which underpin the solution approach here, leverage branch-and-bound and cutting-plane techniques to make large instances tractable. Crucially, MILP's strength lies not only in solution quality but also in the interpretability of its constraint structure, which facilitates direct engagement with operational requirements in a way that black-box simulation cannot easily replicate.

The industrial scalability of MILP-based approaches has been demonstrated in a recent large-scale case study [4], where a mixed-integer formulation solved via column generation across 24 Amazon sites achieved a 14–27% reduction in immobilised working capital alongside service level improvements. While that setting differs from the present one in distribution structure and demand profile, the result is significant: It suggests that the computational cost of MILP does not necessarily prevent its use at an industrial scale when the model is properly extended. Rather than focusing on algorithmic techniques for scaling large problems, this thesis prioritises developing a high-fidelity, high-quality solution for a three-echelon supply chain structure.

### 1.3.3 Spare Parts Inventory in the Automotive Aftermarket

Spare parts inventory in the commercial vehicle sector presents a distinct and underserved class of problems. As [13] documents, performance-based contracts require Original Equipment Manufacturers (OEMs) not only to minimise inventory holding costs but to satisfy hard service level commitments, making service metrics binding constraints rather than secondary objectives. This is a structural feature that distinguishes this domain from general inventory management. What further differentiates spare parts is a combination of demand characteristics that challenge standard inventory models: intermittent, lumpy, or slow-moving demand patterns; extreme stock keeping unit (SKU) diversity with heterogeneous criticality profiles; and long or highly variable lead times [13]. Standard safety stock formulae derived under assumptions of normal, stationary demand are therefore inappropriate without modification, and applying them naively risks both understocking critical items and overstocking slow-movers.

Despite the practical importance of the domain, [2] acknowledges that significant gaps persist between theoretical models and their operational deployment in the automotive industry. Demand forecasting and service parts management remain areas where practice lags well behind what the literature prescribes, in part because most theoretical developments have not been stress-tested against the data irregularities and organisational constraints that characterise real aftermarket environments. The present thesis responds to this gap by grounding its formulation in an industrial automotive dataset.

A further modelling limitation is highlighted by [9], whose work, validated on an industrial automotive dataset, treats lead time as deterministic and explicitly notes that embedding stochastic lead times within the lead-time-demand specification would provide a more

general policy layer. This thesis inherits that limitation: lead times are treated as deterministic, and extending the model to accommodate lead time uncertainty represents a natural and important direction for future work.

## 1.4 Terminology and Key Concepts

The following definitions are based on [1].

**Reorder Point (ROP)** Under a continuous review $(R, Q)$ policy, the reorder point $R$ is the inventory position threshold at which a replenishment order is triggered. For deterministic demand with lead time $L$ and constant demand rate $d$, the reorder point is set to $ROP = Ld$, ensuring that the order arrives exactly when stock is exhausted. Under stochastic demand, the reorder point is set above the expected lead-time demand to provide protection against demand variability.

**Order quantity** The order quantity $Q$ is the fixed batch size placed each time the inventory position reaches or falls below the reorder point. Under the classical Economic Order Quantity (EOQ) model, $Q$ is chosen to balance ordering costs and holding costs. In this thesis, the order quantity $Q_{i,j,t}$ for each item $i$ at each location $j$ and each time period $t$ must be an integer multiple of a sales multiple $q_{i,j}$, i.e. $Q_{i,j,t} = k_{i,j,t} \cdot q_{i,j}$ for some $k_{i,j,t} \in \mathbb{Z}_0^+$.

**Lead time** The lead time $L$ is defined as the total time from the moment an ordering decision is made until the ordered units are available on shelf. This encompasses not only the physical transit time, but also order preparation time, administrative processing time at the supplier, and any inspection time upon receipt. In this model, lead times $L_{i,j}$ are fixed and deterministic.

**Backorder vs. lost sales** When demand cannot be met from stock on hand, two conventions are possible. Under *backordering*, unmet demand is recorded and fulfilled as soon as stock becomes available; the customer waits and the sale is not lost. Under *lost sales*, unmet demand is simply forfeited and the customer goes elsewhere. This model assumes complete backordering: all unmet demand is backordered and carries over to subsequent periods at a fixed cost per unit per period.

**On-hand inventory** The stock on hand $I_{i,j,t}$ is the physical quantity of item $i$ at location $j$ at the end of period $t$ that is available for immediate delivery. It is distinct from the *inventory position*, which additionally accounts for outstanding orders and backorders:

$$\text{Inventory position} = \text{stock on hand} + \text{outstanding orders} - \text{backorders}.$$

On-hand inventory is always non-negative: $I_{i,j,t} \geq 0$.

**Stockout** A stockout occurs when the stock on hand is insufficient to satisfy demand $D_{i,j,t,c}$ in a given period for order class $c$, i.e. when $I_{i,j,t-1} + Q_{i,j,t-L_{i,j}} < D_{i,j,t,c}$. The response to a stockout depends on the echelon at which it occurs.

At the dealer level, a stockout first triggers a rush order (day order or vor depending on the urgency of the demand) placed to the SDC.

At the SDC level, a stockout triggers an order transfer to the CDC. The CDC then fulfills the outstanding order and delivers directly to the dealer.

At the CDC level a stockout results in $I_{i,j,t} = 0$ and a positive backorder quantity $BO_{i,j,t} > 0$, rather than a lost sale.

**Periodic Review Policy** Under periodic review, the inventory position is only observed at fixed, equally spaced points in time separated by a review period $T$. An order may only be placed at these review instances. Compared to continuous review, periodic review requires a larger safety stock, since the inventory position must guard against demand variations over the combined horizon $T + L$ rather than $L$ alone. Periodic review is particularly well suited for coordinating orders across multiple items and for reducing the administrative costs of inventory monitoring.

$(R, Q)$ **Policy** The $(R, Q)$ policy, also referred to as the $(R, nQ)$ policy when multiple $n$ batches may be needed, triggers a replenishment order of size $Q$ (or a multiple thereof) whenever the inventory position declines to or below the reorder point $R$. The two policy parameters, $R$ and $Q$, together determine both the frequency and size of replenishments. In this thesis, the reorder points $ROP_{i,j}$ are the primary decision variables, while order quantities $Q_{i,j,t}$ are constrained to integer multiples of sales multiple $q_{i,j}$.

## 1.5 Research Questions

This thesis is organised around three research questions, each addressing a distinct aspect of the inventory optimisation problem.

The first question is foundational: *how can mathematical optimisation be used to improve spare parts inventory management in a multi-location supply chain?* Volvo's current practice optimises each location independently using a formula-driven single-echelon model. Whether a multi-echelon formulation, one that treats the different warehouses as a coupled system, can produce meaningfully better policies is not obvious, and answering it requires both a rigorous mathematical model and a tractable solution method.

The second question is empirical: *to what extent can a MILP-based approach reduce backorders compared to Volvo Group's current inventory practice?* Formulating a model is not sufficient; it must also outperform the status quo on the metric that matters most operationally. This question grounds the theoretical contribution in a concrete, measurable benchmark.

The third question concerns the structure of the solution space: *what are the trade-offs between inventory cost and service level when optimising reorder points across multiple locations?* A multi-echelon network introduces asymmetries. Holding stock at the CDC is cheap but remote; holding it at the Dealer is expensive but immediately available.

Understanding how the optimiser navigates these asymmetries is as important as the optimal solution itself.

Together, these three questions move from model construction, to validation against practice, to structural insight, the logical arc of the thesis.

## 1.6 Scope and Modelling Assumptions

This BSc thesis was conducted in cooperation with Volvo Group, with the aim of more efficiently minimising backorders across their spare parts supply chain. Currently, Volvo addresses this as a single-echelon problem; this work explores a multi-echelon formulation as a step toward a more integrated approach.

The objective is to determine inventory parameter settings for multiple items across multiple locations that minimise total backorders over a discrete planning horizon measured in days. The supply chain structure considered consists of three echelons: Central Distribution Centres (CDCs), Support Distribution Centres (SDCs), and Dealers. Other warehouse types, such as Regional Distribution Centres (RDCs), are outside the scope of this study.

The following modelling assumptions are considered in this work.

#### Demand and Backorders

- Demand is treated as deterministic in the sense that no probability distribution is assumed; however, it is *not* known in advance. Historical data is used only for post-hoc validation, not as a model input.
- Unmet demand is fully backordered, no lost sales are assumed. Backorders carry over between periods and incur a fixed cost per unit per period.
- Historical data is assumed to be representative of current conditions, i.e. demand patterns observed in the past reflect those of the planning horizon.

The assumption of deterministic demand means that the safety stock decision variable $SS_{i,j}$ is optimised without any stochastic demand term. In a standard $(R, Q)$ formulation under uncertain demand, the reorder point $ROP_{i,j}$ would be set above the expected lead-time demand by a safety factor scaled to demand variance and target service level. Here, because demand in every future period is treated as a known point forecast $\hat{d}_{i,j,t}$, the safety stock collapses to a value determined purely by the constraint structure rather than by any probabilistic buffer calculation.

#### Ordering

- Orders arrive after a fixed, deterministic lead time with no variability. The replenishment fill rate from supplier to CDC is assumed to be 100%.
- Order quantities must be integer multiples of a sales multiple.
- The model operates on calendar days, Monday through Friday.

The assumption that orders arrive after a fixed, deterministic lead time $L_{i,j}$ with no variability, and that supplier fill rates are 100%, similarly suppresses the safety stock term. In the inventory flow constraints at the CDC (2.18), the stock arrival term is subtracted

from on-hand inventory with certainty: there is no probability that the shipment arrives late or in partial quantity. The model therefore never needs to hold precautionary inventory at the CDC to hedge against upstream supply failure; any buffer stock held there arises only when the optimiser is forced to pre-position units to satisfy downstream dealer demand in a later period.

The assumption that order quantities must be integer multiples of a sales multiple $q_i$ is enforced through the integrality constraints on $k^{S}_{i,j,t}$ and $k^{\mathrm{DO}}_{i,j,t}$ (Eq 2.4, 2.5), where $S$ and $DO$ are the different order classes, where $S$ denotes stock orders, and $DO$ denotes rush orders (day orders/VORs). These integrality requirements are precisely what make the model a Mixed-Integer Linear Programme rather than a pure linear one. Relaxing these integrality requirements would yield the LP relaxation discussed in Section 3.3.

**Replenishment Flow**

- Transportation is treated as a black box: only lead times are considered; routing optimisation is out of scope.
- Returns, which may arise when, for example, an expedited replacement order renders an in-transit shipment redundant, are acknowledged as a real phenomenon but are excluded from the formulation.

**Items and Packing**

- Some items may be sold both individually and as part of packs. Each pack, whether a single part or a collection of parts, is treated as an independent stock-keeping unit (SKU), even if its contents overlap with individually sold items.

**Transportation Capacity**

- A single truck type with fixed full-truckload (FTL) volumetric capacity is assumed. Volume is uniform per unit of each item; weight constraints are not considered.
- The number of order lines per location has an upper bound, defined in the model by a fixed threshold, based on real constraints. Each dealer is limited to receiving 1–2 shipments of a given part within any 5-day window, with multiple parts consolidated into a single shipment.

The assumption that a single full-truckload vehicle type governs transportation enters the model as the volumetric capacity constraint in Eq 2.21, which bounds the total shipped volume per distribution centre per period. Weight constraints are excluded by assumption and therefore have no corresponding term in the model. The order-line cap in Eq 2.22 reflects the operational constraint that each dealer can receive at most one or two shipments of a given part within a five-day window, consolidated into a single delivery.

**Costs**

- Holding costs are linear with on-hand inventory. Order costs are class-dependent only, they do not include a one-time charge for placing an order, regardless of how much is ordered. Backorder costs are linear in both quantity and time.

- The unit resale price is fixed and uniform across all locations. Total inventory value must remain within a fixed budget ceiling in every period.

The inventory value cap, derived empirically from each location's historical peak stock value, appears as the constraint in Eq 2.23, and propagates into the safety stock bound of Eq 2.24. Items whose unit price exceeds a location's value budget are excluded from stocking at that location entirely, a rule enforced directly in the constraint structure rather than as a post-processing step.

#### Scope and Representativeness

The set of items and locations selected for this study is assumed to be representative of the broader supply chain. Conclusions drawn from the model are therefore considered generalisable to the full network under the stated assumptions.

## 1.7 Thesis structure

The remainder of this thesis is organised into four chapters.

Chapter 2 develops the mathematical model. It defines the sets, indices, parameters, decision variables, and state variables that constitute the optimisation problem, then presents two objective functions, one targeting service performance and one targeting total cost, together with the full set of operational constraints. Each constraint group is described both formally and in terms of its physical interpretation.

Chapter 3 describes the solution methodology. It provides the necessary background in linear programming, integer programming, and mixed-integer linear programming, and explains the algorithmic techniques, LP relaxation, branch-and-bound, cutting planes, and branch-and-cut, that underpin the Gurobi Optimizer [7] used to solve the model.

Chapter 4 reports numerical results across two problem instances. The first is an academic instance used to examine MILP gap convergence, sensitivity to the big-M parameter, and the effect of the look-ahead window length on solution quality and cost. The second is an industrial instance based on Volvo data, used to compare the model's recommended reorder points and order quantities against Volvo's current operational settings.

Chapter 5 draws conclusions. It analyses the zero-safety-stock pathology that arises under deterministic demand, identifies the key limitations of the present formulation, and outlines a concrete path toward a stochastic extension — including a chance-constrained MILP with normal demand and uncertainty-adjusted order quantities.

# 2. Mathematical Model

This chapter presents the complete mathematical formulation of the inventory optimisation problem. The model is built around a three-echelon network: a Central Distribution Centre (CDC), a Support Distribution Centre (SDC), and a set of dealer nodes, and covers multiple items over a discrete, rolling planning horizon. The chapter is organised as follows. Section 2.1 defines the sets, indices, parameters, and decision and state variables that form the building blocks of the model. Section 2.2 presents the two objective functions: a primary service objective that minimises total backorders, and a cost-based objective that minimises the combined holding, transport, ordering, and shortage costs across the network. Section 2.3 states the full constraint system, and Section 2.4 provides an interpretation of each constraint group, explaining the operational logic it encodes and how it connects to the modelling assumptions of Section 1.6. Taken together, Chapter 2 constitutes a self-contained specification from which the model can be implemented directly in any standard MILP solver.

## 2.1 Sets, Indices and Parameters

### Sets and Indices

| Symbol | Index | Description |
|---|---|---|
| $I$ | $i \in I$ | Set of items, where each item represents a unique stock-keeping unit (SKU). Packs and individually sold parts are treated as distinct items even if their contents overlap. |
| $J$ | $j \in J$ | Set of locations in the supply chain network, comprising the Central Distribution Centre (CDC), the Support Distribution Centre (SDC), and Dealers. Locations are partitioned into $J^{DC} = \{\text{CDC}, \text{SDC}\}$ and $J^{D}$, the set of dealer nodes. |
| $T$ | $t \in T$ | Set of discrete time periods (planning periods), each spanning a fixed number of working days. The model operates on a rolling horizon of $W$ periods. |
| $C$ | $c \in C$ | Set of order channels $C = \{\text{Stock}, \text{Day Orders}, \text{VORs}\}$. *Stock* orders are routine replenishment shipments routed through the supply chain hierarchy; *Rush Orders (Day Orders and VORs)* are emergency replenishments fulfilled directly from the CDC or SDC to a dealer with 1 day lead time for Day Orders, and as fast as possible for VORs. |

Table 2.1: Sets and indices used in the model.

### Decision and State Variables

A distinction is made between *decision variables*: quantities directly controlled by the optimiser; and *state variables*: quantities whose values follow deterministically from the inventory flow conservation constraints once the decision variables are fixed.

The primary decision variable is the safety stock level $SS_{i,j}$, which is held constant over the rolling horizon for each item–location pair. The reorder point is not optimised directly; it is derived from $SS_{i,j}$ as

$$ROP_{i,j} = SS_{i,j} + \widehat{D}^{LT}_{i,j}, \tag{2.1}$$

where $\widehat{D}^{LT}_{i,j} = L_{i,j} \cdot \bar{d}_{i,j}$ is the expected lead-time demand computed from the mean per-period forecast $\bar{d}_{i,j}$.

### Parameters

| Symbol | Domain | Description |
|---|---|---|
| *Decision Variables* | | |
| $SS_{i,j}$ | $\mathbb{R}_0^+$ | Safety stock for item $i$ at location $j$, held constant over the rolling window. The primary decision variable; controls the floor below which on-hand inventory should not fall. |
| $k^S_{i,j,t}$ | $\mathbb{Z}_0^+$ | Shipment multiplier for stock orders of item $i$ to location $j$ in period $t$, such that $Q^S_{i,j,t} = k^S_{i,j,t} \cdot q_i$. |
| $Q^S_{i,j,t}$ | $\mathbb{Z}_0^+$ | Stock order quantity for item $i$ to location $j$ in period $t$. Must be an integer multiple of the sales multiple $q_i$. |
| $k^{DO}_{i,j,t}$ | $\mathbb{Z}_0^+$ | Multiplier for rush (emergency) orders placed by dealer $j \in J^D$ in period $t$. |
| $Q^{DO}_{i,j,t}$ | $\mathbb{Z}_0^+$ | Rush order quantity for item $i$ at dealer $j \in J^D$ in period $t$. Fulfilled immediately (zero lead time). |
| $y_{i,j,t}$ | $\{0,1\}$ | Order trigger indicator; equals 1 if a stock order is placed for item $i$ at location $j$ in period $t$, and 0 otherwise. |
| $z_{i,j,t}$ | $\{0,1\}$ | Rush-order trigger indicator for dealer $j \in J^D$; equals 1 if a DO is triggered in period $t$. |
| *State Variables* | | |
| $I_{i,j,t}$ | $\mathbb{R}_0^+$ | On-hand inventory for item $i$ at location $j$ at the end of period $t$. Equals zero in the event of a stockout at the CDC; equals the positive part of the net inventory balance at dealers and the SDC. |

*Continued on next page*

*Table 2.2 continued from previous page*

| Symbol | Domain | Description |
|---|---|---|
| $BO_{i,t}$ | $\mathbb{R}_0^+$ | Backorders at the CDC for item $i$ at the end of period $t$. Accumulates unmet outbound demand; equals zero when on-hand stock at the CDC is sufficient. |
| $LS_{i,j,t}$ | $\mathbb{R}_0^+$ | Lost sales at dealer $j \in J^D$ for item $i$ in period $t$. Represents dealer demand not satisfied from on-hand stock or rush orders; penalised in the cost objective. |
| $TR_{i,t}$ | $\mathbb{R}_0^+$ | Transfer quantity drawn from the SDC to cover negative net inventory at the SDC in period $t$, passed upstream to the CDC. |

Table 2.2: Decision and state variables. State variables are determined endogenously once the decision variables are fixed.

| Symbol | Description |
|---|---|
| *Demand & Forecast* | |
| $D_{i,j,t}$ | Total demand for item $i$ at location $j$ in period $t$. At the current period $t_{\text{now}}$ this is observed actual demand; for future periods the forecast $\widehat{d}_{i,j,t}$ is used as a proxy. |
| $D^S_{i,j,t}$ | Stock-order component of demand at dealer $j \in J^D$: routine replenishment demand. |
| $D^{DO}_{i,j,t}$ | Rush-order component of demand at dealer $j \in J^D$: emergency demand known at $t_{\text{now}}$; zero for future periods. |
| $\widehat{d}_{i,j,t}$ | Forecast demand for item $i$ at location $j$ in period $t$, used in place of unknown future demand. |
| $\bar{d}_{i,j}$ | Mean per-period forecast demand for item $i$ at location $j$, averaged over the planning horizon: $\bar{d}_{i,j} = T^{-1} \sum_t \widehat{d}_{i,j,t}$. |
| *Cost* | |
| $p_i$ | Unit price of item $i$ (SEK). Used to value inventory and to compute lost-sales at dealers costs. |
| $c_h$ | Holding cost coefficient (fraction of unit price per period). Applied to all on-hand inventory $I_{i,j,t}$. |
| $c_{bw}$ | Backorder penalty per unit at the CDC (SEK per unit per period). |
| $c^{tr}$ | Transport cost per kg for stock shipments (SEK/kg). |
| $c^{rush}$ | Rush transport cost per kg for rush orders (SEK/kg), substantially higher than $c^{tr}$. |

*Continued on next page*

*Table 2.3 continued from previous page*

| Symbol | Description |
|---|---|
| $c_{ol,j}$ | Order-line handling cost at location $j$ (SEK per order line placed). |
| $c_{oh}$ | Order-handling overhead per order line (SEK). |
| $c_{ls}$ | Lost-sales at dealers cost coefficient (multiplied by unit price $p_i$ to obtain the per-unit lost-sales penalty). |
| *Shipment & Transport* | |
| $q_i$ | Sales multiple for item $i$. Order quantities must be integer multiples of $q_i$; partial units are not permitted. |
| $w_i$ | Weight per unit of item $i$ (grams). Used to compute transport costs. |
| $FTL$ | Full truckload volumetric capacity. The total volume of all items shipped to location $j$ in period $t$ may not exceed this limit. |
| $VOL_i$ | Volume per unit of item $i$, uniform across locations. |
| $OL_{th}$ | Maximum number of order lines per shipment, applied as a hard cap. |
| *Lead Time* | |
| $L_{i,j}$ | Lead time for stock orders of item $i$ to location $j$, measured in periods. Fixed and deterministic. Rush orders to dealers carry zero lead time. |
| *Policy & Initialisation* | |
| $M_{i,j}$ | Big-$M$ constant for item $i$ at location $j$. Set item- and location-specifically as $M_{i,j} = 1.2 \cdot \max\Big(\hat{d}_{i,j}^{\max} \cdot 5W,\ 2\,I_{i,j}^{init}\Big)$ to tighten LP relaxations. |
| $IV_{max,j}$ | Maximum total inventory value at location $j$. Derived empirically from the historical peak stock value; may be exceeded in practice. |
| $I_{i,j}^{init}$ | Initial on-hand inventory for item $i$ at location $j$ at the start of the planning horizon. |

Table 2.3: Model parameters grouped by theme.

## 2.2 Objective Functions

The model supports two objective functions. The primary function minimises service failure; the cost-based function enables trade-off analysis between holding costs, transport costs, backorder penalties, and lost sales.

**Primary Objective: Minimise Backorders and Lost Sales**

$$\underset{SS,\,Q^S,\,Q^{DO}\in\mathbb{R}_0^+}{\arg\min} \sum_{i,t} BO_{i,t} \tag{2.2}$$

The first term penalises CDC stockouts; the second penalises any unmet supply commitment to dealers (a feasibility slack); the third is a small regularisation term that discourages unnecessary inventory accumulation without affecting the primary service objective.

**Cost-Based Objective: Minimise Total Operational Cost**

$$\begin{aligned}\min_{SS,\,Q^S,\,Q^{DO}\in\mathbb{R}_0^+} \sum_{t\in W}\Bigg[ & c_h \sum_{i,j} I_{i,j,t}\cdot p_i + c^{tr}\sum_{i,j} Q^S_{i,j,t}\cdot \tfrac{w_i}{1000} + c_{bw}\sum_i BO_{i,t} \\ & + c^{rush}\sum_{i,j\in J^D} Q^{DO}_{i,j,t}\cdot\tfrac{w_i}{1000} + \sum_i\Big(\tfrac{c_{ol,CDC}}{N^{CDC}_{orders}} + c_{oh}\Big)y_{i,CDC,t} \\ & + \sum_i\Big(c_{ol,SDC}+c_{oh}\Big)y_{i,SDC,t} + c_{ls}\sum_{i,j\in J^D} LS_{i,j,t}\cdot p_i\Bigg]\end{aligned} \tag{2.3}$$

The cost components are: holding cost on all on-hand inventory valued at unit price; standard transport cost for stock shipments (weight-based); CDC backorder penalties; rush transport cost for emergency direct orders; order-line handling costs at the CDC; and a lost-sales penalty at dealer level that discourages stockouts and implicitly penalises the need for emergency replenishment. The binary variable $y_{i,j,t}$ indicates whether a positive order is placed thereby activating the corresponding order-line and handling costs. The two objectives are not equivalent: minimising backorders does not minimise total cost, as the cost objective additionally penalises excess inventory, transport frequency, and unmet dealer demand.

## 2.3 Constraints

The constraints translate the physical and operational realities of the supply chain into mathematical boundaries on the decision variables. They govern how orders are sized and triggered, how inventory evolves across echelons in response to demand and replenishment, and what limits apply to transport capacity, order frequency, and stock value. Each constraint group is stated formally below and explained in Section 2.4.

### Sales multiple

$$Q^S_{i,j,t} = k^S_{i,j,t}\cdot q_i, \qquad k^S_{i,j,t}\in\mathbb{Z}_0^+, \qquad \forall\, i,j,t \tag{2.4}$$

$$Q^{DO}_{i,j,t} = k^{DO}_{i,j,t}\cdot q_i, \qquad k^{DO}_{i,j,t}\in\mathbb{Z}_0^+, \qquad \forall\, i,\, j\in J^D,\, t \tag{2.5}$$

### Order trigger linking

$$Q^S_{i,j,t} \le M_{i,j}\cdot y_{i,j,t} \qquad \forall\, i,j,t \tag{2.6}$$

$$q_i\cdot y_{i,j,t} \le Q^S_{i,j,t} \qquad \forall\, i,j,t \tag{2.7}$$

## Dealer inventory flow (stock channel)

Let $I^{pre}_{i,j,t}$ denote the net inventory position at dealer $j \in J^D$ after stock-order arrivals and stock-demand are applied, but before the rush order arrives:

$$I^{pre}_{i,j,t} = I_{i,j,t-1} + Q^{S}_{i,j,\,t-L_{i,j}} - D^{S}_{i,j,t} - D^{DO}_{i,j,t} \qquad \forall\, i,\, j \in J^D,\, t \tag{2.8}$$

$$I_{i,j,t} = \max\Big(0,\ I^{pre}_{i,j,t} + Q^{DO}_{i,j,t}\Big) \qquad \forall\, i,\, j \in J^D,\, t \tag{2.9}$$

The max operator is linearised with the binary variable $\delta_{i,j,t}$ and the item-location-specific big-$M$ constant:

$$I_{i,j,t} \geq 0 \tag{2.10}$$
$$I_{i,j,t} \geq I^{pre}_{i,j,t} + Q^{DO}_{i,j,t} \tag{2.11}$$
$$I_{i,j,t} \leq I^{pre}_{i,j,t} + Q^{DO}_{i,j,t} + M_{i,j}(1 - \delta_{i,j,t}) \tag{2.12}$$
$$I_{i,j,t} \leq M_{i,j}\,\delta_{i,j,t} \tag{2.13}$$

## Rush-order trigger

A rush order is triggered at dealer $j$ when the pre-DO net inventory $I^{pre}_{i,j,t}$ is negative:

$$I^{pre}_{i,j,t} \geq -M_{i,j}\,z_{i,j,t} \qquad \forall\, i,\, j \in J^D,\, t \tag{2.14}$$

$$I^{pre}_{i,j,t} \leq M_{i,j}(1 - z_{i,j,t}) - \varepsilon\, z_{i,j,t} \qquad \forall\, i,\, j \in J^D,\, t \tag{2.15}$$

$$Q^{DO}_{i,j,t} \leq M_{i,j}\,z_{i,j,t} \quad ; \quad Q^{DO}_{i,j,t} \geq q_i\, z_{i,j,t} \qquad \forall\, i,\, j \in J^D,\, t \tag{2.16}$$

## SDC inventory flow

$$I^{raw}_{i,SDC,t} = I_{i,SDC,t-1} + Q^{S}_{i,SDC,\,t-L_{i,SDC}} - \sum_{j \in J^D} Q^{DO}_{i,j,t} \qquad \forall\, i,t \tag{2.17}$$

On-hand SDC inventory and the upstream transfer $TR_{i,t}$ are recovered by the same max-linearisation as for dealers (Eq. 2.10, 2.11, 2.12, 2.13), with $TR_{i,t}$ taking the role of the shortfall.

## CDC inventory flow and backorders

$$I^{raw}_{i,CDC,t} = I_{i,CDC,t-1} + Q^{S}_{i,CDC,\,t-L_{i,CDC}} - \sum_{j \neq CDC} Q^{S}_{i,j,t} - \sum_{j \in J^D} Q^{DO}_{i,j,t} - TR_{i,t} \qquad \forall\, i,t \tag{2.18}$$

CDC backorders arise when $I^{raw}_{i,CDC,t} < 0$ and are linearised analogously to on-hand inventory, using $\delta^{BO}_{i,t}$.

### Safety stock floor

$$I_{i,j,t} \geq SS_{i,j} \qquad \forall\, i, j, t \tag{2.19}$$

### Minimum supply commitment

$$Q^S_{i,j,t} + Q^{DO}_{i,j,t} + I_{i,j,t-1} \geq \widehat{d}_{i,j,t} \qquad \forall\, i,\, j \in J^D,\, t \tag{2.20}$$

### Truck capacity

$$\sum_i Q^S_{i,j,t} \cdot VOL_i \leq FTL \qquad \forall\, j \in J^{DC},\, t \tag{2.21}$$

### Order line cap

$$\sum_i y_{i,j,t} \leq OL_{th} \qquad \forall\, t \tag{2.22}$$

### Inventory value cap

$$\sum_i p_i \cdot I_{i,j,t} \leq IV_{max,j} \qquad \forall\, j,\, t \tag{2.23}$$

### Safety stock value cap

$$SS_{i,j} \leq \frac{IV_{max,j}}{p_i} \qquad \forall\, i, j : p_i \leq IV_{max,j} \tag{2.24}$$

When $p_i > IV_{max,j}$ the item cannot be stocked at location $j$ and $SS_{i,j} = 0$ is enforced directly.

### Non-negativity and initial conditions

$$I_{i,j,t} \geq 0, \quad BO_{i,t} \geq 0, \quad Q^S_{i,j,t} \geq 0, \quad Q^{DO}_{i,j,t} \geq 0, \quad SS_{i,j} \geq 0 \qquad \forall\, i, j, t \tag{2.25}$$

$$I_{i,j,0} = I^{init}_{i,j} \qquad \forall\, i, j \tag{2.26}$$

## 2.4 Constraint Description

**Sales Multiple Rounding (Eq. 2.4, 2.5)** All order quantities, both stock and rush, must be integer multiples of the sales multiple $q_i$. This prevents partial-unit shipments. The multipliers $k^S$ and $k^{DO}$ are the actual integer decision variables; $Q^S$ and $Q^{DO}$ are derived from them.

**Order Trigger Linking (Eq. 2.6, 2.7)** The binary variable $y_{i,j,t}$ is forced to 1 whenever a positive stock order is placed, and to 0 otherwise, via a standard big-M two-sided linking constraint. This allows the order-line cap and order-handling costs to be tracked correctly.

**Two-Channel Dealer Inventory Flow (Eq. 2.8, 2.9)** Dealer inventory evolves through two distinct replenishment channels. The stock channel delivers units ordered $L_{i,j}$ periods earlier; the rush-order channel delivers units within the same period (zero lead time). $I^{pre}_{i,j,t}$ captures the net position after stock arrivals and all demand, before any DO is applied. If $I^{pre}$ is negative, a DO is triggered to cover the shortfall. The max-linearisation (Eq. 2.10, 2.11, 2.12, 2.13) ensures $I_{i,j,t} \geq 0$.

**Rush-Order Trigger (Eq. 2.14, 2.15, 2.16)** The binary variable $z_{i,j,t}$ is set to 1 if and only if $I^{pre}_{i,j,t} < 0$, using a standard big-M complementarity pair (Eq. 2.14, 2.15). When $z = 1$, the linking constraint (Eq. 2.16) forces a DO of at least one sales multiple $q_i$, and the coverage constraint forces the DO to be large enough to eliminate the deficit. The small $\varepsilon$ in Eq. 2.15 ensures strict complementarity.

**SDC and CDC Inventory Flow (Eq. 2.17, 2.18)** The SDC balance deducts all rush orders dispatched to dealers in period $t$: the SDC acts as the emergency fulfilment hub. The CDC balance deducts all outbound stock shipments to the SDC and to dealers, all rush orders, and any upstream transfer $TR_{i,t}$ triggered when the SDC net balance is negative. CDC backorders accumulate when outbound commitments exceed available stock.

**Safety Stock Floor (Eq. 2.19)** On-hand inventory at every location is constrained to remain at or above $SS_{i,j}$ in every period. This is the mechanism by which the optimised safety stock is enforced; it replaces the classical reorder-point trigger and makes the order decisions implicit rather than explicit.

**Minimum Supply Commitment (Eq. 2.20)** This constraint ensures that the combined incoming supply and current inventory at each dealer covers the forecast demand in every period.

**Truck Capacity and Order Line Cap (Eq. 2.21, 2.22)** The truck capacity constraint limits total shipment volume per DC per period to the full truckload capacity $FTL$. The order line cap limits the total number of stock order shipments dispatched across all locations within a given planning period.

**Inventory Value Cap (Eq. 2.23, 2.24)** Total inventory value at each location is capped at $IV_{max,j}$, derived from the historical peak stock value. The safety stock is additionally bounded by $\frac{IV_{max,j}}{p_i}$ per item to prevent the optimiser from recommending a safety stock level that alone would exceed the location's value budget. Items priced above the location budget are excluded from stocking at that location entirely.

**Non-negativity and Initial Conditions (Eq. 2.25, 2.26)** All state and decision variables are non-negative. The planning horizon is initialised with observed on-hand inventory $I^{init}_{i,j}$; no initial backorders are assumed.

# 3. Solution Methodology

The methodological approach taken in this thesis is centred on Mixed-Integer Linear Programming (MILP), a branch of mathematical optimisation suited to problems that combine continuous decision variables with discrete or integer-valued ones. MILP is well-established in supply chain and inventory management literature as a means of optimising stocking policies when real-world constraints (such as rounding, truck capacity, and periodic review cycles) introduce integrality requirements that render purely continuous methods inadequate.

The overall methodology proceeds in three stages. First, the inventory replenishment problem is formalised mathematically: the relevant sets, decision variables, state variables, parameters, objective functions, and constraints are defined precisely. This formalisation translates the physical problem of determining reorder points and order quantities across a multi-item, multi-location network into a structured optimisation problem amenable to algorithmic solution. Second, the resulting MILP is solved using Gurobi Optimizer [7], applied to both a simplified and a more complex problem instance in order to assess solution quality, computational tractability, and sensitivity to problem scale. Third, the optimised reorder points $R_{i,j,t}$ and order quantities $Q_{i,j,t}$ are compared against the benchmark parameters currently used by Volvo, and evaluated against historical demand data in a post-hoc validation step. Because demand is treated as deterministic within the model and historical data serve only as validation input, this step assesses how well the optimised parameters would have performed against observed demand patterns. Unlike simulation-based or heuristic approaches, the MILP framework provides provably optimal solutions, or bounded approximations thereof, with respect to the stated objective and constraints. The principal trade-off is that computational tractability depends on both problem size and formulation strength.

The definitions and ideas in the following sections are based on [10].

## 3.1 Linear Programming

An optimisation problem seeks to find values of a set of decision variables that minimise or maximise an objective function subject to a set of constraints. When the objective function and all constraints are linear and decision variables are continuous, the problem is a Linear Program (LP).

The objective function takes the form:

$$f(x_1, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

and constraints are expressed as:

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b$$

Constraints define the operational and physical limitations under which the system must operate. Together, the objective function and constraints determine the feasible region. For inventory management problems, constraints typically represent inventory balance equations, replenishment rules, lead-time relationships, and storage limitations.

In the model developed in this thesis, the objective is to minimise total accumulated backorders across all items, locations, and time periods over the planning horizon. The objective function is linear in the backorder state variables, and inventory flow conservation constraints govern the evolution of inventory levels over time.

## 3.2 Integer and Mixed-Integer Linear Programming

The definitions and ideas in this section are based on [12].

### 3.2.1 Mixed-Integer Linear Programming

Many real-world optimisation problems involve decision variables that must take discrete values, for example, numbers of vehicles, production batches, or inventory replenishment quantities. When all variables are restricted to integer values, the problem is an Integer Linear Program (ILP):

$$\begin{aligned} \max \quad & c^\top x \\ & Ax \leq b \\ & x \geq 0 \text{ and integer} \end{aligned}$$

where $A$ is an $m \times n$ matrix, $c$ an $n$-dimensional row vector, $b$ an $m$-dimensional column vector, and $x$ an $n$-dimensional column vector of integer variables. When a model contains both continuous and integer-valued decision variables, it is a Mixed-Integer Linear Program (MILP):

$$\begin{aligned} \max \quad & c^\top x + h^\top y \\ & Ax + Gy \leq b \\ & x \geq 0, \quad y \geq 0 \text{ and integer} \end{aligned}$$

where $G$ is $m \times p$, $h$ is a $p$-dimensional row vector, and $y$ is a $p$-dimensional column vector of integer variables.

The inventory optimisation problem considered in this thesis is formulated as a MILP, as described in Chapter 2. The decision variables $ROP_{i,j} \in \mathbb{Z}_0^+$ and $k_{i,j,t} \in \mathbb{Z}_0^+$ are integer-valued, reflecting the discrete nature of replenishment quantities and threshold-based reorder policies. The state variables $I_{i,j,t}$ and $BO_{i,j,t}$ are determined entirely by the inventory flow conservation constraints once the decision variables are fixed, and are therefore not independently controlled by the optimiser.

### 3.2.2 Binary Variables

When all variables are restricted to $\{0, 1\}$ values, the problem is a Binary Integer Program:

$$\begin{aligned} \max \quad & c^\top x \\ & Ax \leq b \\ & x \in \{0,1\}^n \end{aligned}$$

Binary variables are introduced to model conditional logic that cannot be represented within a purely linear framework. In the proposed model, an order is placed only when the projected inventory position at the end of the preceding period, $IP_{i,j,t-1}$, falls at or below the reorder point. This condition is represented using the binary trigger variable $y_{i,j,t}$ and enforced through the Big-$M$ linearisation described in Equation 2.6. When the reorder condition is satisfied, $y_{i,j,t}$ is forced to one, permitting a positive replenishment quantity; otherwise no order may be placed.

The introduction of binary variables creates a large number of possible ordering schedules, and as the numbers of items, locations, and planning periods increase, the number of feasible combinations grows rapidly, contributing significantly to computational complexity.

## 3.3 Solution Methods

**LP Relaxation** A key concept in solving MILPs is the LP relaxation, obtained by removing all integrality requirements and allowing integer variables to take continuous values. For example, a binary variable $y \in \{0,1\}$ is relaxed to $0 \leq y \leq 1$.

For the integer program $\max\{cx : x \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$, the LP relaxation is $z^{LP} = \max\{cx : x \in P\}$.

The LP relaxation is solvable efficiently and its objective value provides an upper bound on the optimal MILP solution, forming the foundation for exact solution methods. To illustrate, consider the integer program:

$$z = \max \; 4x_1 + 2x_2$$

subject to

$$\begin{aligned} 3x_1 + x_2 &\leq 11, \\ x_1 + 2x_2 &\leq 8, \\ x_2 &\leq 3, \\ x &\in \mathbb{Z}^2_+. \end{aligned}$$

The point $(x_1, x_2) = (3, 2)$ is feasible with objective value $z = 4(3) + 2(2) = 16$, giving the lower bound $z \geq 16$. The LP relaxation optimum occurs at the intersection of $3x_1 + x_2 = 11$ and $x_1 + 2x_2 = 8$, yielding $x_1 = 14/5$, $x_2 = 13/5$, with LP objective value $z^{LP} = 4(14/5) + 2(13/5) = 82/5 = 16.4$. Since the optimal integer objective must be integral, $z \leq \lfloor 16.4 \rfloor = 16$, and combining bounds gives $z^* = 16$, attained at $(x_1, x_2) = (3, 2)$.

**Branch-and-Bound** Branch-and-bound is the primary exact algorithm for solving MILPs. The method begins by solving the LP relaxation of the original problem. If the solution satisfies all integrality requirements, it is also optimal for the MILP. Otherwise,

the solver selects a fractional variable and creates two subproblems by imposing a floor bound on one and a ceiling bound on the other, forming nodes of a search tree. LP relaxations are solved at each node, and nodes are *pruned* when their bound is worse than the best known integer solution, when infeasible, or when they already yield an integer-feasible solution. This process continues until optimality is proven or a stopping criterion is reached.

**Cutting Planes** To improve the efficiency of branch-and-bound, cutting planes, valid inequalities that remove fractional solutions from the LP relaxation without excluding any feasible integer solutions, are often added. By tightening the LP relaxation, cutting planes improve bound quality and can significantly reduce the number of nodes that must be explored.

**Branch-and-Cut** Modern MILP solvers combine both techniques within a branch-and-cut framework: LP relaxations are solved throughout the search tree while cutting planes are generated dynamically at fractional nodes. This produces stronger bounds, reduces tree size, and generally leads to substantially faster solution times than pure branch-and-bound.

## 3.4 Gurobi Optimizer

**Solver Overview** Gurobi Optimizer [7] is a state-of-the-art commercial solver for LPs, MILPs, and quadratic optimisation models, widely used in both academia and industry. The inventory optimisation model in this thesis is implemented in Python via the `gurobipy` interface.

**Presolve** Before the main optimisation procedure, Gurobi applies presolve techniques that simplify the model by eliminating redundant variables and constraints, tightening variable bounds, aggregating constraints, and identifying infeasible or dominated regions of the search space. For large-scale instances, presolve can substantially reduce problem size and improve computational performance.

**Branch-and-Cut** Gurobi solves MILPs through the branch-and-cut framework described above. The quality of the current best feasible solution is continuously compared against the best available bound, and the process terminates when the gap falls below a predefined tolerance or another stopping criterion is met.

**Heuristics** In addition to exact techniques, Gurobi employs heuristic procedures to identify high-quality feasible solutions early in the search process. These incumbent solutions improve pruning efficiency within the branch-and-cut tree and often contribute substantially to reductions in overall solution time, though they do not alone guarantee optimality.

**Parallelisation** Gurobi exploits multi-core architectures through parallel computation, processing multiple nodes of the search tree simultaneously. This capability is particularly important for the inventory optimisation problem considered here, where the number of decision variables and constraints grows substantially with the number of items, locations, and planning periods.

# 4. Numerical Results

## 4.1 Problem Instances

Two problem instances are considered. The first is a simplified instance involving a small number of items and locations, intended to verify the model formulation and provide interpretable baseline results. The second is a larger industrial instance reflecting the scale and structure of Volvo's spare parts network, used to assess computational performance and the quality of the resulting inventory policies.

## 4.2 Academic Problem Instance

### 4.2.1 Overview

This section reports the numerical analysis of three aspects of the rolling-horizon MIP formulation described in Chapter 2: MIP gap convergence across rolling-horizon periods (Panel A), sensitivity of solution quality to the big-$M$ parameter (Panel B), and the effect of the look-ahead window length $W$ on total cost and solve time (Panel C). The instance comprises two items, one CDC, one SDC, and one dealer, solved over a five-period horizon $T = \{M1, \ldots, M5\}$ under Objective 2 (cost minimisation). All experiments use Gurobi 13.0.1 with `MIPGap = 0.1%` and `TimeLimit = 300 s`.

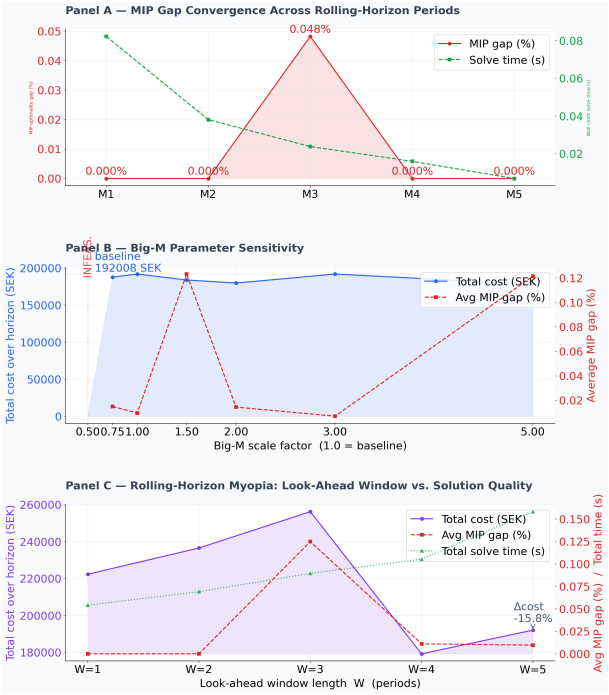


Figure 4.1: Computational Validation and Sensitivity Analysis of the Multi-Item Multi-Location Inventory Optimization Model.

### 4.2.2 Panel A — MIP Gap Convergence Across Rolling-Horizon Periods

Panel A of Figure 4.1 plots the MIP optimality gap (left axis, red) and wall-clock solve time (right axis, green) at each of the five rolling-horizon periods M1–M5.

Four of the five periods achieve a 0.000% MIP gap, confirming that the branch-and-bound search terminates at the root node. The single exception is period M3, which records a gap of 0.048% (still two orders of magnitude below the 0.1% stopping criterion) before returning to zero at M4 and M5. Solve times decline monotonically from approximately 0.06 s at M1 to under 0.005 s at M5.

Both patterns are consistent with the structural properties of the rolling-horizon formulation. As the remaining horizon contracts, the feasible region shrinks: fewer binary trigger variables $y_{i,j,t}$ and $z_{i,j,t}$ remain undecided, the big-$M$ constraints (Eq. 2.14, 2.15, 2.16) become tighter relative to the active solution, and the LP relaxation bound closes rapidly on the MILP optimal. The anomaly at M3 likely reflects a period in which forecast demand $\widehat{d}_{i,j,t}$ and the current pipeline produce a binding minimum-supply constraint (Eq. 2.20), temporarily expanding the combinatorial search before the solver recovers.

The warm-start procedure, in which the optimal solution from period $t$ is used to initialise the solver at period $t+1$, is the primary explanation for the steep solve-time decline. As committed pipeline quantities are fixed via the `pipeQ` and `pipeDO` constraints, the effective number of free variables decreases each period, and the warm start increasingly provides a near-feasible starting point.

### 4.2.3 Panel B — Big-$M$ Parameter Sensitivity

Panel B examines how scaling the big-$M$ constants $M_{i,j}$ affects total cost and average MILP gap across all five periods, with the scale factor ranging from 0.50 to 5.00 relative to the baseline value (1.0).

Total cost (blue, left axis) is largely stable across the range $[0.75, 5.00]$, fluctuating narrowly around 175,000–190,000, with the exception of a sharp spike to approximately 192,000 at scale factor 1.50. The average MILP gap (red dashed, right axis) is near zero for most scale factors but exhibits two pronounced peaks: a large spike to approximately 0.12% at scale factor 1.50, and a moderate increase above 0.10% at scale factor 5.00.

These results are consistent with known theoretical properties of big-$M$ formulations. If M is too small (scale factor below 1.0), the complementarity constraints in Eqs. 2.14 and 2.15 may not deactivate correctly, potentially causing infeasibility. This occurs when $M_{i,j}$ is smaller than the actual net inventory $I^{pre}_{i,j,t}$. The scale factor of 0.50 avoids a cost spike only because the small problem instance happens to keep $I^{pre}$ within range; this would not generalise. An oversized $M$ (scale factor 5.00 and above) weakens the LP relaxation: the feasible region defined by constraints such as

$$I_{i,j,t} \leq I^{raw}_{i,j,t} + M_{i,j}(1 - \delta_{i,j,t})$$

becomes looser, so the LP bound retreats further from the MILP optimal and the branch-and-bound tree widens. The spike at scale factor 1.50 is anomalous and suggests a regime boundary at which the $M$ value is large enough to weaken the relaxation but not large enough to dominate all feasible inventory trajectories, maximising numerical

ill-conditioning. The baseline value of $M_{i,j} = 1.2 \times \max(\hat{d}_{i,j}^{\max} \cdot 5W,\ 2\, I_{i,j}^{init})$ is positioned in the stable flat region and keeps the average MIP gap below 0.02% while maintaining feasibility.

### 4.2.4 Panel C — Rolling-Horizon Myopia: Look-Ahead Window vs. Solution Quality

Panel C varies the look-ahead window length $W \in \{1, 2, 3, 4, 5\}$ and records total cost over the horizon (blue, left axis), average MIP gap (red dashed, right axis), and total solve time (green dotted, right axis).

Total cost exhibits a non-monotone pattern. Starting from approximately 221,000 at $W = 1$, cost rises to 238,000 at $W = 2$, spikes to 255,000 at $W = 3$, then drops sharply to a minimum of approximately 181,000 at $W = 4$ before recovering slightly to 192,000 at the full horizon $W = 5$. The net reduction from the myopic baseline to the best window is $\Delta\text{cost} = -15.8\%$, annotated directly on the figure.

The intermediate cost spike at $W = 3$ is accompanied by the highest average MILP gap across all window lengths, approaching 0.13%. This is consistent with the horizon myopia effect: a window of three periods is long enough to require the solver to commit to replenishment decisions whose downstream consequences interact with the safety-stock floor (Eq. 2.19) and minimum-supply constraints (Eq. 2.20) in ways that a shorter window avoids by ignoring future periods and a longer window resolves by seeing the full demand forecast. At $W = 3$ the solver must balance holding costs against potential future backorder penalties without sufficient look-ahead to determine the correct timing of replenishment orders, producing both higher cost and a larger integrality gap.

Total solve time increases approximately linearly with $W$, as expected: each additional period adds one time slice of binary variables $y_{i,j,t}$ and $z_{i,j,t}$ and their associated big-$M$ constraints. Solve time reaches approximately 0.09 s at $W = 5$, which is well within the 300 s time limit and confirms that the computational cost of extending the window is negligible for this instance size.

The result that $W = 4$ dominates $W = 5$ on cost is consistent with end-of-horizon effects: the full-horizon solve at $W = 5$ must simultaneously determine replenishment decisions for all five periods, and the terminal periods carry no salvage value for residual inventory, inducing late-horizon under-ordering. A window of $W = 4$ avoids this by always maintaining one unseen future period that implicitly motivates the solver to hold buffer stock, acting as a soft proxy for the missing continuation value.

### 4.2.5 Summary of Results

Table 4.1 summarises the key findings from the three panels.

| Experiment | Key finding | Optimal setting | Risk |
|---|---|---|---|
| MIP gap convergence | Gap $\leq$ 0.048% at all periods; solve time falls with horizon | — | Mild spike at M3 |
| Big-$M$ sensitivity | Cost stable for scale $\in [0.75, 3.00]$; gap spikes at 1.50× and 5.00× | 1.0× baseline | Ill-conditioning at 1.5× |
| Window length | $W = 4$ minimises cost (−15.8% vs. myopic); $W = 3$ is worst | $W = 4$ | Cost spike + high gap at $W = 3$ |

Table 4.1: Summary of analytical results across the three experiments.

Taken together, the analytical results confirm three properties of the formulation. First, the rolling-horizon MILP solves efficiently to near-optimality at every period for this instance size, with the warm-start and pipeline-fixing mechanisms eliminating almost all combinatorial complexity in later periods. Second, the big-$M$ calibration is consequential: the baseline formula produces stable, low-gap solutions, while both under- and over-scaling introduce numerical pathologies. Third, the look-ahead window introduces a non-trivial myopia–complexity trade-off: a window of $W = 4$ achieves the best cost outcome, but $W = 3$ is strictly dominated and should be avoided.

## 4.3 Industrial Problem Instance

The following subsections present a systematic analysis of the optimisation output across five result figures. Each figure compares Volvo's current operational parameters against those produced by the Mixed-Integer Linear Programme (MILP) under two objectives: **Obj 1** (backorder minimisation) and **Obj 2** (total cost minimisation). The discussion proceeds from the reorder-point and economic-order-quantity comparisons at two representative locations, the Central Distribution Centre (CDC) and Dealer 5, to the temporal inventory-flow trajectories at the CDC.

### 4.3.1 Reorder Point at the CDC — Period M4

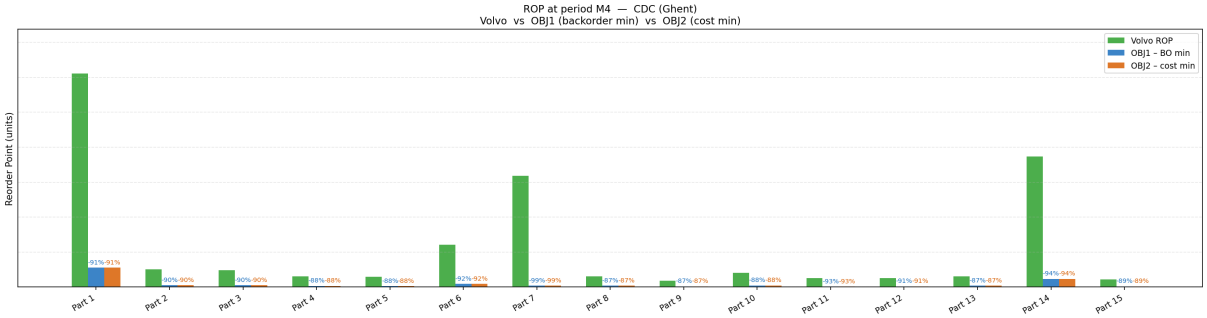


Figure 4.2: Reorder Point (ROP) at period M4 for the CDC: Volvo's current setting (green) versus the MILP under Obj 1 – backorder minimisation (blue) and Obj 2 – cost minimisation (orange). Percentage annotations indicate the relative change from Volvo's ROP.

Figure 4.2 reveals a systematic and substantial divergence between Volvo's current reorder points and those recommended by either objective of the MILP. For Parts 1, 7, and 14 both Obj 1 and Obj 2 recommend reductions of 87–94 % across every part for which a comparison is possible. Crucially, the blue (Obj 1) and orange (Obj 2) bars are nearly indistinguishable throughout: the ROP compression is not a cost-cutting artefact but reflects the genuinely optimal replenishment trigger under *both* objectives.

**Structural reasons for the divergence.** The ROP in this model is defined as

$$\mathrm{ROP}_{i,j,t} = \mathrm{SS}_{i,j,t} + L_j \cdot \bar{d}_i, \tag{4.1}$$

where $\mathrm{SS}_{i,j,t}$ is the safety stock for item $i$ at location $j$ in period $t$, $L_j$ is the fixed lead time at location $j$, and $\bar{d}_i$ is the mean forecast demand for item $i$ averaged over all planning periods. Two structural assumptions of the MILP directly suppress the ROP relative to Volvo's formula.

First, the model assumes *100 % supplier availability* and a *perfectly known, fixed lead time* on the upstream (supplier-to-CDC) arc. Safety stock exists precisely to absorb lead-time and supply uncertainty; when those sources of uncertainty are removed by assumption, the safety stock term $\mathrm{SS}_{i,j,t}$ collapses to a value far below what a real-world logistics formula would produce. Volvo's operational ROP is calibrated to hedge against supplier variability, shipment delays, and forecast errors that are not modelled here. The large green bars in Figure 4.2 therefore represent not over-stocking per se but rather the real-world risk premium embedded in Volvo's parameters.

Second, the lead-time demand component $L_j \cdot \bar{d}_i$ in (4.1) is *period-invariant*: it depends only on the location and the item, not on the specific planning period. This is a modelling simplification that prevents the ROP from adapting to seasonal or trend patterns within the horizon; in practice, if demand is higher in certain periods, a dynamic ROP formula would respond accordingly. This rigidity is a recognised limitation of the formulation.

**Interpretation.** The near-zero MILP ROPs indicate that, given the large initial inventory at the CDC and the deterministic demand forecasts, the optimiser can satisfy all demand without triggering replenishment at the CDC level during much of the horizon. This does not imply that the MILP recommends eliminating replenishment entirely; rather, it shifts the replenishment logic from threshold-based triggers to explicit period-by-period order decisions embedded in the MILP constraints. In a real environment, relaxing the perfect-supply assumption would increase the model's CDC ROP, though likely still below, Volvo's current values.

### 4.3.2 Reorder Point at Dealer 5 — Period M4

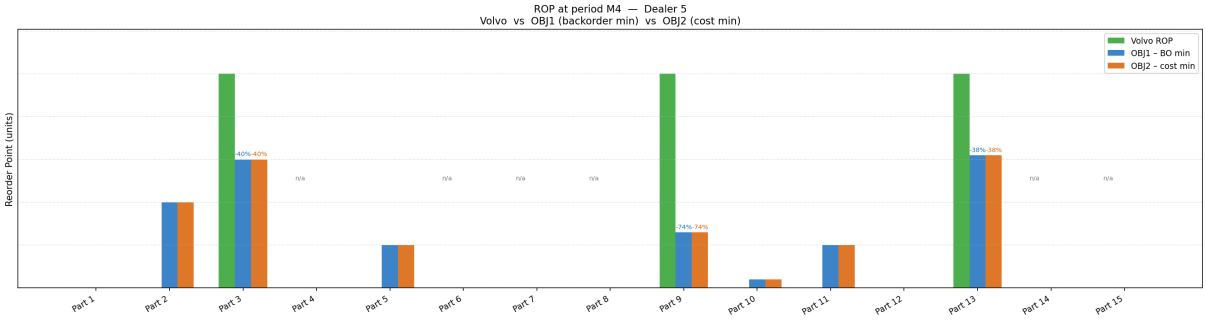


Figure 4.3: Reorder Point (ROP) at period M4 for Dealer 5: Volvo's current setting (green) versus Obj 1 (blue) and Obj 2 (orange). Parts labelled `n/a` have no Volvo historical ROP on record; percentage annotations show the relative change where comparison is possible.

At the dealer level the picture changes considerably. Volvo's own ROPs at Dealer 5 are already small reflecting the low-volume, high-variety nature of dealer demand for automotive spare parts. The MILP reduces these further: $-40\,\%$ for Part 3, $-74\,\%$ for Part 9, and $-38\,\%$ for Part 13. For parts where Volvo records no ROP (labelled `n/a`), the model produces small negative or near-zero values, which arise because the formula in (4.1) can yield a negative ROP when the safety stock is zero and the lead-time demand is negligible; operationally, a negative ROP is floored at zero, meaning no replenishment trigger exists for those items.

Importantly, both objectives again produce identical or nearly identical dealer ROPs, confirming that the reduction is driven by the structure of dealer demand and the assumed supply chain rather than by the cost objective. Because dealer demand volumes are very low relative to the CDC, the lead-time demand term in (4.1) contributes little, and the safety stock under deterministic forecasts is also small. The magnitude of the gap between Volvo's and the MILP's recommendations is therefore much narrower at the dealer than at the CDC, simply because Volvo's own dealer ROPs were already near the lower bound.

### 4.3.3 Economic Order Quantity at the CDC — Period M4

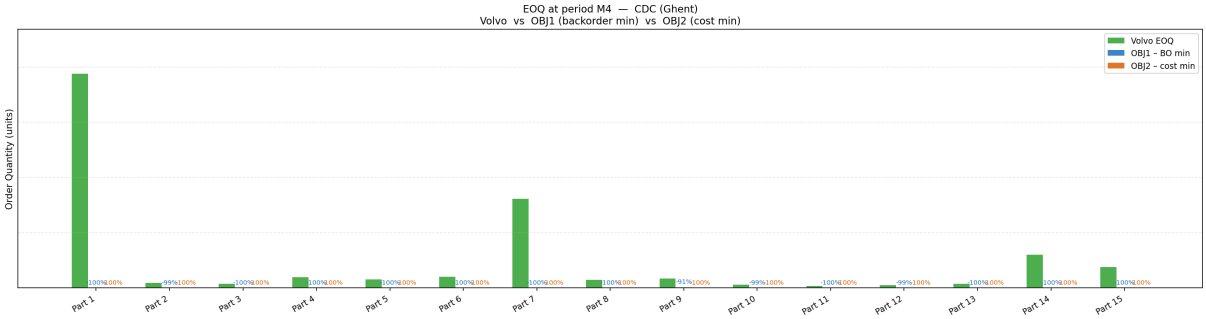


Figure 4.4: Economic Order Quantity (EOQ) at period M4 for the CDC: Volvo's current setting (green) versus Obj 1 (blue) and Obj 2 (orange). Percentage annotations indicate the relative change from Volvo's EOQ.

Figure 4.4 shows the EOQ comparison at the CDC. Volvo's values are highest for Part 1 and Part 7, while the MILP recommends reductions of 99–100 % for virtually every part under both objectives. The pattern is again symmetric across Obj 1 and Obj 2, implying that the EOQ compression is not a consequence of the cost minimisation trade-off but a structural feature of the optimisation environment.

**Structural reasons for the EOQ divergence.** The classical EOQ formula

$$Q^* = \sqrt{\frac{2\,K\,\lambda}{h}}, \tag{4.2}$$

where $K$ is the fixed ordering cost, $\lambda$ is the demand rate, and $h$ is the holding cost per unit per period, balances ordering frequency against holding cost. Large batch sizes (high $Q^*$) arise when ordering costs are large relative to holding costs, or when demand volume is high. Volvo's EOQ at the CDC reflects both of these factors under real-world conditions.

In the MILP, the same assumptions that suppress the ROP also affect the EOQ. Because the supplier delivers to the CDC with 100 % availability and a known fixed lead time, the model can schedule arbitrarily many individual replenishment events across the planning horizon without risk of stock-out between the order placement and its arrival. There is therefore no incentive to consolidate demand into large batches as a hedge against supply uncertainty; the optimiser instead places small, precisely timed orders that match per-period demand exactly. Furthermore, the MILP formulates order quantities as continuous decision variables subject to demand constraints, which allows batch sizes approaching zero, a flexibility unavailable in practice due to minimum order quantities, rounding rules, and logistics constraints that the model does not encode.

The combined effect is that Volvo's historically large CDC batch sizes, which implicitly reflect supply risk, consolidation economies, and minimum order agreements, appear drastically oversized relative to the model's frictionless replenishment environment.

**Interpretation.** The near-zero MILP EOQs should not be interpreted as a recommendation to place infinitesimally small orders. Rather, they reflect the model's ability to replace

a fixed-batch reorder policy with an explicit, period-by-period order schedule optimised over the full horizon. The practical implication is that a model incorporating realistic order-size constraints, such as a minimum order quantity or rounding, would produce batch sizes closer to Volvo's current settings.

### 4.3.4 Economic Order Quantity at Dealer 5 — Period M4

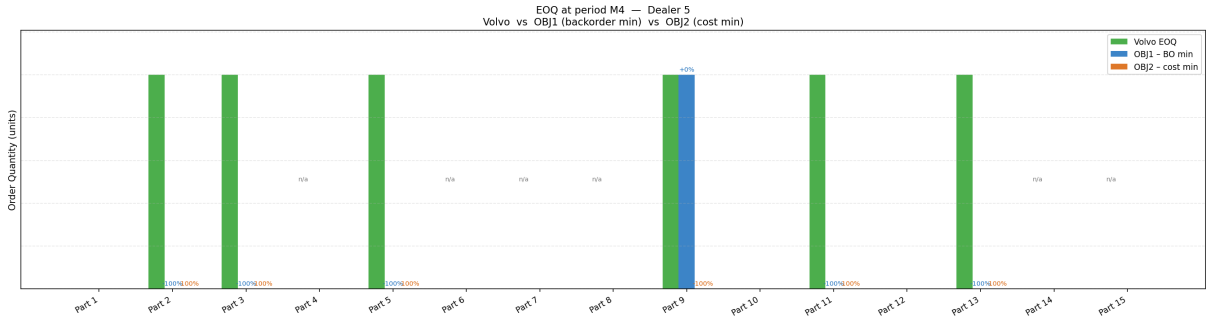


Figure 4.5: Economic Order Quantity (EOQ) at period M4 for Dealer 5: Volvo's current setting (green) versus Obj 1 (blue) and Obj 2 (orange). Parts labelled `n/a` have no Volvo historical EOQ on record.

At Dealer 5 the EOQ picture mirrors the ROP finding. Volvo's dealer EOQs are already at or near 1 unit for Parts 2, 3, 5, 9, 11, and 13, reflecting the low-volume, unit-level replenishment typical of an automotive spare-parts dealer. The MILP either matches this value exactly or reduces it to zero (a −100 % reduction).

The single noteworthy exception is Part 9 under Obj 1, annotated +0 %: the MILP independently arrives at an EOQ of 1 unit, identical to Volvo's own setting. This agreement is not coincidental; for very low-demand items where the optimal batch size is trivially 1 unit per replenishment event, both the heuristic and the optimiser converge to the same solution, providing a point of external validation for the MILP output at the dealer level.

Under Obj 2, the EOQ for Part 9 is also driven to zero (or effectively zero), consistent with the cost minimiser's preference for delaying replenishment and exhausting existing stock rather than triggering new orders. The absence of Obj 2 bars for several parts (displayed as `n/a`) indicates that Volvo holds no EOQ record for those item-dealer combinations, so no comparative annotation is possible.

### 4.3.5 Inventory Flow at the CDC Over Time


Figure 4.6: Inventory flow at the CDC over the 5-period planning horizon for all 15 parts. The solid blue line shows CDC inventory under Obj 1 (backorder minimisation); the dashed orange line shows Obj 2 (cost minimisation). Grey bars indicate actual realised demand in each period. The secondary $y$-axis (right) shows demand in units.

Figure 4.6 provides the richest picture of the optimisation dynamics, decomposing the inventory trajectory at the CDC across all 15 parts and 5 periods.

**Objective 1 (backorder minimisation).** Under Obj 1, several parts exhibit a monotonically increasing or sustained CDC inventory despite consumption by demand. Parts 2, 4, 8, 10, 12, and 15 show the blue line rising or remaining elevated throughout the horizon. This behaviour arises because Obj 1 penalises backorders with no offsetting holding-cost penalty: the optimiser stocks inventory aggressively at the CDC to guarantee downstream supply, and it does not face any financial deterrent against accumulating buffer stock. The result is that CDC inventory can grow well beyond what demand requires, particularly for parts where the initial inventory is already substantial.

**Objective 2 (cost minimisation).** Obj 2 exhibits a qualitatively different trajectory. For the majority of parts (most clearly Parts 5 and 6) the orange dashed line begins at a relatively high level and declines steadily over the horizon as existing stock is consumed. Replenishment is triggered sparingly, and only when the cost of a backorder exceeds the combined cost of ordering and holding. This "exhaust-first" strategy is the direct consequence of the holding-cost term in Obj 2's objective function, which penalises unnecessary stock retention.

**Early-period cost absorption.** For several parts (notably Parts 1 and 5), both inventory curves start from a low position relative to the first-period demand bar. This reflects backorders carried into the planning horizon from before period M11, or unusually high M11 demand relative to initial stock. The optimiser absorbs these early-period penalties and then operates efficiently for the remainder of the horizon; the cumulative cost curve is consequently front-loaded, as previously discussed.

**Parts with negligible CDC inventory.** Parts 7, 9, and 13 show both objectives maintaining near-zero CDC inventory throughout. For these parts, the optimiser relies either on the System Distribution Centre (SDC) or on existing dealer stock to fulfil demand, bypassing the CDC entirely. This is consistent with the near-zero ROP and EOQ values observed for those parts in Figures 4.2 and 4.4.

**Divergence between objectives.** The largest divergence between the two objectives occurs for high-demand, high-initial-inventory parts. For Part 1 the gap between the blue and orange lines widens progressively over the horizon: Obj 1 sustains approximately 400–500 units of CDC inventory through period M5 before declining, while Obj 2 depletes from a comparable starting point to near zero by M5. For Part 6 the divergence is even more pronounced: the blue line sustains inventory above 1 000 units into the later periods, whereas the orange line falls toward zero by M5 and remains negligible thereafter. These parts therefore represent the clearest instances where the choice of objective function has a material operational consequence.

### 4.3.6 Summary of Key Findings

Table 4.2 consolidates the principal findings from the five figures.

Table 4.2: Summary of ROP and EOQ comparison between Volvo's current settings and the MILP at the CDC and Dealer 5 (period M4). Percentage ranges indicate the reduction recommended by the model (identical for Obj 1 and Obj 2 in all cases).

| **Parameter** | **Location** | **Volvo range** | **Model reduction** |
|---|---|---|---|
| ROP | CDC (Ghent) | 2 000 – 61 000 units | 87–94 % |
| ROP | Dealer 5 | 0–1 unit | 38–74 % |
| EOQ | CDC (Ghent) | 100 – 7 700 units | 99–100 % |
| EOQ | Dealer 5 | 0–1 unit | $\leq$ 100 % |

Three overarching conclusions emerge from this analysis.

1. **Perfect-supply assumptions drive parameter compression.** The single most important factor explaining the gap between Volvo's operational parameters and the MILP recommendations is the model's assumption of 100 % supplier availability with a fixed, known lead time on the upstream arc. This removes the uncertainty premium that Volvo's logistics formulae embed in both the ROP and the EOQ. In a stochastic extension of the model, where lead-time variability and supplier fill rates are modelled explicitly, the recommended ROP and EOQ values would increase toward Volvo's current settings.

2. **The MILP objectives differ primarily in their CDC inventory behaviour.** At the dealer level and in the ROP/EOQ parameters, Obj 1 and Obj 2 produce nearly identical recommendations. The material difference emerges in the CDC inventory trajectories (Figure 4.6), where the absence of a holding-cost penalty in Obj 1 allows unconstrained stock accumulation, while Obj 2 enforces a lean, demand-paced depletion policy.

3. **Model limitations constrain operational applicability.** The period-invariant lead-time demand term in the ROP formula (Equation (4.1)), the absence of minimum order quantities, and the deterministic demand representation each contribute to parameters that are structurally lower than what real-world operations require. These limitations are important to acknowledge when interpreting the percentage reductions in Figures 4.2, 4.3, 4.4, 4.5 as policy recommendations rather than exact operational targets.

# 5. Conclusion

## 5.1 Summary of Findings

This thesis developed a MILP formulation for jointly optimising reorder points and order quantities across a three-echelon spare parts network (CDCs, SDCs, and Dealers) over a discrete rolling horizon. Three research questions were addressed.

**Can mathematical optimisation improve multi-echelon spare parts inventory management?** Yes. The formulation encodes operational constraints (such as rounding, full-truckload capacity, order-line caps, and inventory value ceilings) directly into the model, producing policies that are both mathematically optimal and operationally coherent. Solve times remain well below one second per period, with MILP gaps below 0.05%, confirming computational tractability at the scales tested.

**Can a MILP reduce backorders relative to Volvo's current practice?** The model produces reorder points 87–94% lower than Volvo's CDC parameters and order quantities near zero, however a gap not necessarily evidence of systematic over-stocking. Under deterministic demand, the theoretically correct safety stock is zero: there is no uncertainty for it to absorb. Volvo's operational parameters include forecast error, shipment delays, and supplier variability, none of which are present in the model. A rigorous comparison requires a stochastic extension that places both approaches in the same informational environment.

**What are the trade-offs between inventory cost and service level?** The two objectives differ at the CDC for high-demand, high-initial-inventory parts. Backorder minimisation (Objective 1) accumulates buffer inventory while cost minimisation (Objective 2) pursues lean depletion, driving CDC stock towards zero by the end of the horizon. The latter is capital-efficient, but leaves no buffer against demand spikes or supply disruptions. Neither objective alone is sufficient for operational implementation; a weighted or chance-constrained formulation that balances holding costs against a service-level target is required.

The zero-safety-stock outcome is therefore not a modelling flaw but instead highlights the difference between the model's recommendations and Volvo's current parameters. This difference reflects the extra buffer, or uncertainty premium, built into Volvo's logistics system through its reorder points and order quantities.

## 5.2 Future Work

The most important limitation is the deterministic demand assumption, and the most direct remedy is a chance-constrained MILP with stochastic demand. Concretely, this would replace the point-forecast demand $d_{i,j,t}$ with its mean and standard deviation estimated from historical order data, introduce analytically derived safety stock parameterised by a target service level $\alpha$ per echelon, add lead-time variance on the supplier-to-CDC arc, and

adjust EOQ lower bounds using the uncertainty-corrected formula

$$Q^*_{\text{stoch}} = \sqrt{\frac{2K(\lambda + \sigma_d^2/\lambda)}{h}}, \tag{5.1}$$

which recovers the classical result when $\sigma_d^2 = 0$ and prevents the near-zero batch sizes observed in the industrial results. This formulation remains a standard MILP (with second-order cone constraints if chance constraints are not linearised) and requires no structural changes to the solver pipeline.

For intermittent-demand spare parts, Sample Average Approximation (SAA) provides a natural framework in which sampled stochastic programs are solved repeatedly and statistical estimates are used to construct confidence intervals for solution quality, as mentioned in [3].

When the number of scenarios becomes large, [3] also talks about decomposition methods such as the L-shaped (Benders) method to exploit the stochastic structure and avoid solving the full extensive form directly. For multistage settings, aggregation of periods or rolling-horizon approximations can mitigate the exponential growth in problem size associated with increasing numbers of stages and realizations.

Finally, integration with Volvo's system would allow the model to consume live demand signals and delivery records, closing the loop between optimisation and execution.

# Bibliography


[1] Sven Axsäter. *Inventory Control*. 3rd ed. Vol. 225. International Series in Operations Research and Management Science. Springer, 2015. ISBN: 978-3-319-15728-3.

[2] Sendhil Nathan Balasubramanian et al. "Innovative Framework for Effective Service Parts Management in the Automotive Industry". In: (Dec. 2023). DOI: 10.20944/preprints202312.1051.v1.

[3] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer, 1997. ISBN: 0-387-98217-5.

[4] Marco Caserta and Luca D'Angelo. "Optimising two-echelon spare parts inventory: a case study from Amazon's operations". In: *International Journal of Production Research* 0.0 (2026), pp. 1–19. DOI: 10.1080/00207543.2026.2655026. eprint: https://doi.org/10.1080/00207543.2026.2655026. URL: https://doi.org/10.1080/00207543.2026.2655026.

[5] Ayse Sena Eruguz et al. "A comprehensive survey of guaranteed-service models for multi-echelon inventory optimization". In: *International Journal of Production Economics* 172 (2016), pp. 110–125. ISSN: 0925-5273. DOI: https://doi.org/10.1016/j.ijpe.2015.11.017. URL: https://www.sciencedirect.com/science/article/pii/S0925527315005162.

[6] João NC Gonçalves, M Sameiro Carvalho, and Paulo Cortez. "Operations Research Models and Methods for Safety Stock Determination: A Review". In: 7 (2020). DOI: 10.1016/j.orp.2020.100164.

[7] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2026. URL: https://www.gurobi.com.

[8] Xu Li, Xiaoheng Ji, and Xiaolong Zeng. "Optimizing supply chain networks using mixed integer linear programming (MILP)". In: *Theoretical and Natural Science* 53 (Sept. 2024), pp. 10–15. DOI: 10.54254/2753-8818/53/20240642.

[9] Dimitrios S. Sfiris and Dimitrios E. Koulouriotis. "A New Approach to Forecast Intermittent Demand and Stock-Keeping-Unit Level Optimization for Spare Parts Management". In: *Applied Sciences* 15.22 (2025). ISSN: 2076-3417. DOI: 10.3390/app152212030. URL: https://www.mdpi.com/2076-3417/15/22/12030.

[10] Robert J. Vanderbei. *Linear Programming*. 4th ed. Vol. 196. International Series in Operations Research and Management Science. Springer. ISBN: 978-1-4614-7629-0.

[11] Joaquim Jorge Vicente. "Optimizing Supply Chain Inventory: A Mixed Integer Linear Programming Approach". In: *Systems* 13.1 (2025). ISSN: 2079-8954. DOI: 10.3390/systems13010033. URL: https://www.mdpi.com/2079-8954/13/1/33.

[12] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley  Songs, 1998.

[13] Shuai Zhang, Kai Huang, and Yufei Yuan. "Spare Parts Inventory Management: A Literature Review". In: *Sustainability* 13.5 (2021). ISSN: 2071-1050. DOI: 10.3390/su13052460. URL: https://www.mdpi.com/2071-1050/13/5/2460.

# Appendix: Code

```
import math
import sys
import time
from collections import defaultdict
from typing import Dict, List, Tuple

import gurobipy as gp
from gurobipy import GRB

ITEMS     = ["PartA", "PartB"]
PRICE     = {"PartA": 500.0,  "PartB": 1_200.0}
WEIGHT    = {"PartA": 800.0,  "PartB": 2_500.0}
VOLUME    = {"PartA": 0.002,  "PartB": 0.008}
LOT_SIZE = {"PartA": 1, "PartB": 1}

LOCATIONS = ["CDC", "SDC", "D1"]
DEALERS   = ["D1"]

LEAD_TIME = {
    ("supplier", "CDC"): 2,
    ("CDC",      "SDC"): 1,
    ("CDC",      "D1"):  1,
    ("CDC/SDC",  "D1"):  0,
}

N_PERIODS = 5
PERIODS   = list(range(1, N_PERIODS + 1))
WINDOW    = PERIODS

D_ACTUAL: Dict[Tuple[str, str, int], float] = {
    ("PartA", "D1", 1): 8.0,  ("PartA", "D1", 2): 10.0, ("PartA",
        "D1", 3): 9.0,
    ("PartA", "D1", 4): 11.0, ("PartA", "D1", 5): 10.0,
    ("PartB", "D1", 1): 3.0,  ("PartB", "D1", 2): 4.0,  ("PartB",
        "D1", 3): 3.0,
    ("PartB", "D1", 4): 5.0,  ("PartB", "D1", 5): 4.0,
    **{(i, "CDC", t): 0.0 for i in ITEMS for t in PERIODS},
    **{(i, "SDC", t): 0.0 for i in ITEMS for t in PERIODS},
}

FC: Dict[Tuple[str, str, int], float] = {
    ("PartA", "D1", 1): 9.0,  ("PartA", "D1", 2): 9.0,  ("PartA",
        "D1", 3): 10.0,
    ("PartA", "D1", 4): 10.0, ("PartA", "D1", 5): 10.0,
    ("PartB", "D1", 1): 3.0,  ("PartB", "D1", 2): 4.0,  ("PartB",
        "D1", 3): 4.0,
    ("PartB", "D1", 4): 4.0,  ("PartB", "D1", 5): 4.0,
```

```
    **{(i, "CDC", t): 0.0 for i in ITEMS for t in PERIODS},
    **{(i, "SDC", t): 0.0 for i in ITEMS for t in PERIODS},
}

I_INIT: Dict[Tuple[str, str], float] = {
    ("PartA", "CDC"): 25.0, ("PartA", "SDC"): 10.0, ("PartA", "D1"): 5.0,
    ("PartB", "CDC"): 12.0, ("PartB", "SDC"):  4.0, ("PartB", "D1"): 2.0,
}

BO_INIT = {"PartA": 0.0, "PartB": 0.0}

# Cost parameters     actual values are proprietary and have been omitted.
# Values should be set according to the specific network being modelled.
C_HOLDING = None
TRANSPORT_KG = None
RUSH_KG = None
C_BACKORDER = None
C_ORDERLINE_CDC = None
C_ORDERLINE_SDC = None
C_ORDERHANDLING = None
C_LOST_SALES = None
CDC_ORDERS_PER_PERIOD = None

FTL_CAPACITY  = 1e9
OL_THRESHOLD  = max(len(ITEMS) * 2, 30)
INV_VALUE_MAX = {"CDC": 5_000_000.0, "SDC": 800_000.0, "D1": 200_000.0}

EPSILON = 1e-4

LT_LOC = {
    "CDC": LEAD_TIME[("supplier", "CDC")],
    "SDC": LEAD_TIME[("CDC", "SDC")],
    "D1":  LEAD_TIME[("CDC", "D1")],
}

MEAN_FC: Dict[Tuple[str, str], float] = {
    (i, j): sum(FC.get((i, j, t), 0.0) for t in PERIODS) / N_PERIODS
    for i in ITEMS for j in LOCATIONS
}

LT_DEMAND: Dict[Tuple[str, str], float] = {
    (i, j): LT_LOC[j] * MEAN_FC[(i, j)]
    for i in ITEMS for j in LOCATIONS
}

```

```
def _compute_mbig(scale=1.0):
    M_BIG: Dict[Tuple[str, str], float] = {}
    for i in ITEMS:
        for j in LOCATIONS:
            peak_d = max((D_ACTUAL.get((i, j, t), 0.0) for t in
               PERIODS), default=0.0)
            peak_f = max((FC.get((i, j, t), 0.0)       for t in
               PERIODS), default=0.0)
            peak   = max(peak_d, peak_f, 1.0)
            init   = I_INIT.get((i, j), 0.0)
            M_BIG[(i, j)] = max(peak * N_PERIODS * 5, init * 2) *
                1.2 * scale
    return M_BIG

M_BIG = _compute_mbig(scale=1.0)

print("=" * 65)
print("  ACADEMIC INVENTORY MODEL")
print("=" * 65)
print(f"  Items: {ITEMS}  |  Locations: {LOCATIONS}  |  Periods:
   {PERIODS}")


def lead_time_for(j, emergency=False):
    if emergency and j in DEALERS:
        return LEAD_TIME[("CDC/SDC", j)]
    if j == "CDC":
        return LEAD_TIME[("supplier", "CDC")]
    if j == "SDC":
        return LEAD_TIME[("CDC", "SDC")]
    return LEAD_TIME[("CDC", j)]


def stock_order_period(j, arrival_t):
    return arrival_t - lead_time_for(j)


OBJ_BACKORDER = "backorder"
OBJ_COST      = "cost"


def build_milp(t_now, window, I_current, BO_current,
               pipeline_stock, pipeline_do, objective_mode=
                  OBJ_COST,
               m_big_override=None, collect_mip_log=None):
    """
    collect_mip_log: if a list is passed, incumbent MIP gap/time
       pairs are appended
                     via a Gurobi callback.
    m_big_override:  dict with same keys as M_BIG; used for
       sensitivity experiments.
```

```
    """
    M = m_big_override if m_big_override is not None else M_BIG

    tau_min     = window[0]
    dealers_set = set(DEALERS)

    comm_stock_arr, comm_stock_ord = {}, {}
    comm_do_arr,    comm_do_ord    = {}, {}

    for (i, j, arr_t), qty in pipeline_stock.items():
        if arr_t in window and qty > 0:
            comm_stock_arr[(i, j, arr_t)] = qty
            comm_stock_ord[(i, j, stock_order_period(j, arr_t))]
               = qty

    for (i, j, arr_t), qty in pipeline_do.items():
        if arr_t in window and qty > 0 and j in dealers_set:
            comm_do_arr[(i, j, arr_t)] = qty
            comm_do_ord[(i, j, arr_t - LEAD_TIME[("CDC/SDC", j)])
               ] = qty

    m = gp.Model(f"Acad_{objective_mode}_t{t_now}")
    m.Params.OutputFlag = 0          # suppress per-solve output
       in sensitivity runs
    m.Params.MIPGap     = 0.01
    m.Params.TimeLimit  = 120.0

    SS     = m.addVars(ITEMS, LOCATIONS,          lb=0.0,
                       name="SS")
    k      = m.addVars(ITEMS, LOCATIONS, window,  vtype=GRB.
       INTEGER, lb=0, name="k")
    Q      = m.addVars(ITEMS, LOCATIONS, window,  lb=0.0,
                       name="Q")
    k_DO   = m.addVars(ITEMS, DEALERS,   window,  vtype=GRB.
       INTEGER, lb=0, name="k_DO")
    Q_DO   = m.addVars(ITEMS, DEALERS,   window,  lb=0.0,
                       name="Q_DO")
    I_raw  = m.addVars(ITEMS, LOCATIONS, window,  lb=-GRB.INFINITY
       ,  name="I_raw")
    Ipos   = m.addVars(ITEMS, LOCATIONS, window,  lb=0.0,
                       name="I")
    BO     = m.addVars(ITEMS,            window,  lb=0.0,
                       name="BO")
    LS     = m.addVars(ITEMS, DEALERS,   window,  lb=0.0,
                       name="LS")
    TR     = m.addVars(ITEMS,            window,  lb=0.0,
                       name="TR")
    y      = m.addVars(ITEMS, LOCATIONS, window,  vtype=GRB.BINARY
       ,  name="y")
    z      = m.addVars(ITEMS, DEALERS,   window,  vtype=GRB.BINARY
       ,  name="z")
```

```
    delta     = m.addVars(ITEMS, LOCATIONS, window, vtype=GRB.
        BINARY, name="delta")
    delta_BO = m.addVars(ITEMS,            window, vtype=GRB.
        BINARY, name="dBO")
    delta_TR = m.addVars(ITEMS,            window, vtype=GRB.
        BINARY, name="dTR")
    supply_slack = m.addVars(ITEMS, DEALERS, window, lb=0.0, name
        ="supply_slack")

    for (i, j, ord_t), qty in comm_stock_ord.items():
        if ord_t in window:
            m.addConstr(Q[i, j, ord_t] == qty)
    for (i, j, ord_t), qty in comm_do_ord.items():
        if ord_t in window and j in dealers_set:
            m.addConstr(Q_DO[i, j, ord_t] == qty)

    for i in ITEMS:
        qi = LOT_SIZE[i]
        for j in LOCATIONS:
            for tau in window:
                if (i, j, tau) not in comm_stock_ord:
                    m.addConstr(Q[i, j, tau] == k[i, j, tau] * qi
                        )
        for d in DEALERS:
            for tau in window:
                if (i, d, tau) not in comm_do_ord:
                    m.addConstr(Q_DO[i, d, tau] == k_DO[i, d, tau
                        ] * qi)

    for i in ITEMS:
        qi = LOT_SIZE[i]
        for j in LOCATIONS:
            Mi = M[(i, j)]
            for tau in window:
                m.addConstr(Q[i, j, tau] <= Mi * y[i, j, tau])
                m.addConstr(Q[i, j, tau] >= qi * y[i, j, tau])

    def stock_arrival(i, j, tau):
        e = gp.LinExpr(0.0)
        if (i, j, tau) in comm_stock_arr:
            e += comm_stock_arr[(i, j, tau)]
        else:
            ord_t = stock_order_period(j, tau)
            if ord_t in window and (i, j, ord_t) not in
                comm_stock_ord:
                e += Q[i, j, ord_t]
            elif ord_t < window[0]:
                e += pipeline_stock.get((i, j, tau), 0.0)
        return e

    def do_arrival(i, d, tau):
```

```
213             e = gp.LinExpr(0.0)
214             if (i, d, tau) in comm_do_arr:
215                 e += comm_do_arr[(i, d, tau)]
216             else:
217                 ord_t = tau if tau == tau_min else tau - 1
218                 if ord_t in window and (i, d, ord_t) not in
                        comm_do_ord:
219                     e += Q_DO[i, d, ord_t]
220             return e
221
222         I_pre_vars = {}
223
224         for i in ITEMS:
225             for tau in window:
226                 def prev_I(j, _tau=tau, _i=i):
227                     return I_current.get((_i, j), 0.0) if _tau ==
                            tau_min else Ipos[_i, j, _tau - 1]
228
229                 for d in DEALERS:
230                     Mi  = M[(i, d)]
231                     dem = D_ACTUAL.get((i, d, tau), 0.0) if tau ==
                            t_now else FC.get((i, d, tau), 0.0)
232                     I_pre = m.addVar(lb=-GRB.INFINITY, name=f"I_pre_{
                            i}_{d}_{tau}")
233                     I_pre_vars[(i, d, tau)] = I_pre
234                     m.addConstr(I_pre == prev_I(d) + stock_arrival(i,
                             d, tau) - dem)
235                     m.addConstr(I_raw[i, d, tau] == I_pre +
                            do_arrival(i, d, tau))
236                     m.addConstr(LS[i, d, tau] >= dem - Ipos[i, d, tau
                            ])
237                     m.addConstr(I_pre >= -Mi * z[i, d, tau])
238                     m.addConstr(I_pre <= Mi * (1 - z[i, d, tau]) -
                            EPSILON * z[i, d, tau])
239                     m.addConstr(Ipos[i, d, tau] >= 0)
240                     m.addConstr(Ipos[i, d, tau] >= I_raw[i, d, tau])
241                     m.addConstr(Ipos[i, d, tau] <= I_raw[i, d, tau] +
                             Mi * (1 - delta[i, d, tau]))
242                     m.addConstr(Ipos[i, d, tau] <= Mi * delta[i, d,
                            tau])
243
244                 Mi_s = M[(i, "SDC")]
245                 m.addConstr(I_raw[i, "SDC", tau] == prev_I("SDC") +
                        stock_arrival(i, "SDC", tau)
246                                 - gp.quicksum(Q_DO[i, d, tau] for d in
                                    DEALERS))
247                 m.addConstr(Ipos[i, "SDC", tau] >= 0)
248                 m.addConstr(Ipos[i, "SDC", tau] >= I_raw[i, "SDC",
                        tau])
249                 m.addConstr(Ipos[i, "SDC", tau] <= I_raw[i, "SDC",
                        tau] + Mi_s * (1 - delta[i, "SDC", tau]))
```

```
250             m.addConstr(Ipos[i, "SDC", tau] <= Mi_s * delta[i, "
                    SDC", tau])
251             m.addConstr(TR[i, tau] >= 0)
252             m.addConstr(TR[i, tau] >= -I_raw[i, "SDC", tau])
253             m.addConstr(TR[i, tau] <= -I_raw[i, "SDC", tau] +
                    Mi_s * (1 - delta_TR[i, tau]))
254             m.addConstr(TR[i, tau] <= Mi_s * delta_TR[i, tau])
255             m.addConstr(I_raw[i, "SDC", tau] >= -Mi_s * delta_TR[
                    i, tau])
256             m.addConstr(I_raw[i, "SDC", tau] <= Mi_s * (1 -
                    delta_TR[i, tau]) - EPSILON * delta_TR[i, tau])
257
258             Mi_c = M[(i, "CDC")]
259             m.addConstr(I_raw[i, "CDC", tau] == prev_I("CDC") +
                    stock_arrival(i, "CDC", tau)
260                         - gp.quicksum(Q[i, d, tau] for d in
                            DEALERS)
261                         - Q[i, "SDC", tau] - TR[i, tau])
262             m.addConstr(BO[i, tau] >= 0)
263             m.addConstr(BO[i, tau] >= -I_raw[i, "CDC", tau])
264             m.addConstr(BO[i, tau] <= -I_raw[i, "CDC", tau] +
                    Mi_c * (1 - delta_BO[i, tau]))
265             m.addConstr(BO[i, tau] <= Mi_c * delta_BO[i, tau])
266             m.addConstr(I_raw[i, "CDC", tau] >= -Mi_c * delta_BO[
                    i, tau])
267             m.addConstr(I_raw[i, "CDC", tau] <= Mi_c * (1 -
                    delta_BO[i, tau]) - EPSILON * delta_BO[i, tau])
268             m.addConstr(Ipos[i, "CDC", tau] >= 0)
269             m.addConstr(Ipos[i, "CDC", tau] >= I_raw[i, "CDC",
                    tau])
270             m.addConstr(Ipos[i, "CDC", tau] <= I_raw[i, "CDC",
                    tau] + Mi_c * (1 - delta[i, "CDC", tau]))
271             m.addConstr(Ipos[i, "CDC", tau] <= Mi_c * delta[i, "
                    CDC", tau])
272
273     for i in ITEMS:
274         for j in LOCATIONS:
275             if PRICE[i] > INV_VALUE_MAX[j]:
276                 m.addConstr(SS[i, j] == 0.0)
277                 continue
278             m.addConstr(SS[i, j] <= INV_VALUE_MAX[j] / PRICE[i])
279             for tau in window:
280                 m.addConstr(Ipos[i, j, tau] >= SS[i, j])
281
282     for i in ITEMS:
283         for d in DEALERS:
284             for tau in window:
285                 fc_d = FC.get((i, d, tau), 0.0)
286                 if fc_d > 1e-6:
287                     prev_inv = (Ipos[i, d, tau - 1] if tau >
                            window[0]
```

```
                                else I_current.get((i, d), 0.0))
                    m.addConstr(Q[i, d, tau] + Q_DO[i, d, tau] +
                        prev_inv
                                + supply_slack[i, d, tau] >= fc_d
                                    )

    for i in ITEMS:
        qi = LOT_SIZE[i]
        for d in DEALERS:
            if PRICE[i] > INV_VALUE_MAX[d]:
                continue
            for tau in window:
                order_t = tau - LEAD_TIME[("CDC/SDC", d)]
                if (i, d, order_t) not in comm_do_ord and order_t
                    in window:
                    if (i, d, tau) not in I_pre_vars:
                        continue
                    I_pre = I_pre_vars[(i, d, tau)]
                    Mi    = M[(i, d)]
                    m.addConstr(Q_DO[i, d, order_t] >= -I_pre -
                        Mi * (1 - z[i, d, tau]))
                    m.addConstr(Q_DO[i, d, order_t] <= Mi * z[i,
                        d, tau])
                    m.addConstr(Q_DO[i, d, order_t] >= qi * z[i,
                        d, tau])

    for j in ("CDC", "SDC"):
        for tau in window:
            m.addConstr(gp.quicksum(Q[i, j, tau] * VOLUME[i] for
                i in ITEMS) <= FTL_CAPACITY)
            ol_cap = OL_THRESHOLD * CDC_ORDERS_PER_PERIOD if j ==
                "CDC" else OL_THRESHOLD
            m.addConstr(gp.quicksum(y[i, j, tau] for i in ITEMS)
                <= ol_cap)
    for j in LOCATIONS:
        for tau in window:
            m.addConstr(gp.quicksum(Ipos[i, j, tau] * PRICE[i]
                for i in ITEMS) <= INV_VALUE_MAX[j])

    if objective_mode == OBJ_BACKORDER:
        obj = (gp.quicksum(BO[i, tau] for i in ITEMS for tau in
            window)
                + gp.quicksum(supply_slack[i, d, tau]
                              for i in ITEMS for d in DEALERS for
                                  tau in window)
                + 1e-4 * gp.quicksum(Ipos[i, j, tau] * PRICE[i]
                                     for i in ITEMS for j in
                                         LOCATIONS for tau in window
                                     ))
    else:
        obj = gp.LinExpr()
```

```
325         for tau in window:
326             obj += C_HOLDING * gp.quicksum(Ipos[i, j, tau] *
                    PRICE[i]
327                                            for i in ITEMS for j
                                                 in LOCATIONS)
328             obj += TRANSPORT_KG * gp.quicksum(Q[i, j, tau] *
                    WEIGHT[i] / 1_000.0
329                                               for i in ITEMS for
                                                    j in LOCATIONS)
330             obj += C_BACKORDER * gp.quicksum(BO[i, tau] for i in
                    ITEMS)
331             obj += RUSH_KG * gp.quicksum(Q_DO[i, d, tau] * WEIGHT
                    [i] / 1_000.0
332                                          for i in ITEMS for d in
                                                DEALERS)
333             obj += gp.quicksum(
334                 (C_ORDERLINE_CDC / CDC_ORDERS_PER_PERIOD +
                        C_ORDERHANDLING) * y[i, "CDC", tau]
335                 + (C_ORDERLINE_SDC + C_ORDERHANDLING) * y[i, "SDC
                        ", tau] for i in ITEMS)
336             obj += C_LOST_SALES * gp.quicksum(LS[i, d, tau] *
                    PRICE[i]
337                                               for i in ITEMS for
                                                    d in DEALERS)
338             obj += C_BACKORDER * gp.quicksum(supply_slack[i, d,
                    tau]
339                                              for i in ITEMS for d
                                                  in DEALERS)
340
341     m.setObjective(obj, GRB.MINIMIZE)
342     return m, dict(SS=SS, k=k, Q=Q, Q_DO=Q_DO, I=Ipos, I_raw=
            I_raw,
343                    BO=BO, TR=TR, y=y, z=z, LS=LS,
344                    supply_slack=supply_slack, window=window,
                           t_now=t_now)
345
346
347 def rolling_horizon_solve(objective_mode=OBJ_COST, m_big_override
        =None,
348                           fixed_horizon=None, verbose=True):
349     """
350 ␣␣␣␣fixed_horizon:␣if␣an␣int␣W␣is␣given,␣every␣period␣uses␣a␣
        window␣of␣exactly␣W␣steps
351 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(truncated␣to␣remaining␣periods).␣␣␣␣used␣for␣
        horizon-length␣sensitivity.
352 ␣␣␣␣m_big_override:␣function␣(scale)␣->␣M_BIG␣dict,␣or␣a␣ready-
        made␣dict.
353 ␣␣␣␣Returns␣extended␣result␣dict␣including␣per-period␣MIP␣gap␣&␣
        solve␣time.
354 ␣␣␣␣"""
355     if verbose:
```

```
356         label = objective_mode.upper()
357         print(f"\n{'#'*65}\n␣␣ROLLING␣HORIZON␣   ␣Objective:␣{
                label}\n{'#'*65}")
358
359     inv_value_ts, SS_ts, EOQ_ts, ROP_ts, cost_ts, ls_ts = {}, {},
            {}, {}, {}, {}
360     mip_gap_ts  = {}   # period -> MIP gap %
361     solve_time_ts = {}  # period -> wall-clock seconds
362     pipeline_stock:  Dict[Tuple[str, str, int], float] = {}
363     pipeline_do:     Dict[Tuple[str, str, int], float] = {}
364     Q_executed:      Dict = defaultdict(float)
365     Q_DO_executed:   Dict = defaultdict(float)
366     I_current  = {(i, j): float(I_INIT.get((i, j), 0.0)) for i in
            ITEMS for j in LOCATIONS}
367     BO_current = {i: float(BO_INIT.get(i, 0.0)) for i in ITEMS}
368
369     for t in PERIODS:
370         if verbose:
371             print(f"\n[{objective_mode.upper()}]␣Period␣t␣=␣{t}")
372         if t > 1:
373             for i in ITEMS:
374                 for d in DEALERS:
375                     arrivals = (pipeline_stock.get((i, d, t),
                                0.0)
376                                 + pipeline_do.get((i, d, t), 0.0)
                                    )
377                     I_current[(i, d)] = max(
378                         0.0, I_current.get((i, d), 0.0) +
                                arrivals
379                         - D_ACTUAL.get((i, d, t - 1), 0.0))
380                 sdc_arr    = pipeline_stock.get((i, "SDC", t),
                            0.0)
381                 sdc_do_out = sum(Q_DO_executed.get((i, d, t - 1),
                            0.0) for d in DEALERS)
382                 I_current[(i, "SDC")] = max(0.0, I_current.get((i
                            , "SDC"), 0.0)
383                                             + sdc_arr -
                                                sdc_do_out)
384                 cdc_arr    = pipeline_stock.get((i, "CDC", t),
                            0.0)
385                 cdc_out    = sum(Q_executed.get((i, j, t - 1),
                            0.0)
386                                  for j in LOCATIONS if j != "CDC"
                                     )
387                 cdc_do_out = sum(Q_DO_executed.get((i, d, t - 1),
                            0.0) for d in DEALERS)
388                 I_current[(i, "CDC")] += cdc_arr - cdc_out -
                            cdc_do_out
389
390         for pipe in (pipeline_stock, pipeline_do):
391             for key in [k for k in pipe if k[2] <= t]:
```

```
392                 del pipe[key]
393
394         if fixed_horizon is None:
395             window = list(range(t, N_PERIODS + 1))
396         else:
397             window = list(range(t, min(t + fixed_horizon,
                    N_PERIODS + 1)))
398
399         t0 = time.perf_counter()
400         m, h = build_milp(t_now=t, window=window, I_current=
                I_current,
401                           BO_current=BO_current,
402                           pipeline_stock=pipeline_stock,
                                pipeline_do=pipeline_do,
403                           objective_mode=objective_mode,
404                           m_big_override=m_big_override)
405         m.optimize()
406         solve_time_ts[t] = time.perf_counter() - t0
407
408         if m.Status == GRB.INFEASIBLE:
409             m.computeIIS()
410             for c in m.getConstrs():
411                 if c.IISConstr:
412                     print(f"  IIS: {c.ConstrName}", file=sys.
                            stderr)
413             raise RuntimeError(f"Infeasible at t={t}.")
414         if m.Status not in (GRB.OPTIMAL, GRB.TIME_LIMIT, GRB.
                SUBOPTIMAL):
415             raise RuntimeError(f"Solve failed at t={t}, status={m
                    .Status}.")
416
417         try:
418             cost_ts[t] = m.ObjVal
419             # MIP gap: 0 if optimal (status 2), else m.MIPGap
420             mip_gap_ts[t] = m.MIPGap * 100.0  # as percent
421         except gp.GurobiError:
422             cost_ts[t]    = float("nan")
423             mip_gap_ts[t] = float("nan")
424
425         SS_v, k_v, Qv, Q_DOv, Ipos, BO_v = h["SS"], h["k"], h["Q"
                ], h["Q_DO"], h["I"], h["BO"]
426         LS_v = h.get("LS")
427
428         for i in ITEMS:
429             try:
430                 BO_current[i] = BO_v[i, t].X
431             except (gp.GurobiError, KeyError):
432                 BO_current[i] = 0.0
433
434             if objective_mode == OBJ_COST and LS_v is not None:
435                 ls_total = sum(
```

```
                    LS_v[i, d, t].X if (i, d, t) in LS_v.keys()
                        else 0.0
                    for d in DEALERS)
                ls_ts[(i, t)] = ls_total

            for j in LOCATIONS:
                try:
                    inv_val = Ipos[i, j, t].X * PRICE[i]
                except (gp.GurobiError, KeyError):
                    inv_val = 0.0
                inv_value_ts[(i, j, t)] = inv_val

                try:
                    ss_val  = SS_v[i, j].X
                    eoq_val = k_v[i, j, t].X * LOT_SIZE[i]
                except (gp.GurobiError, KeyError):
                    ss_val = eoq_val = 0.0
                SS_ts[(i, j, t)]  = ss_val
                EOQ_ts[(i, j, t)] = eoq_val
                ROP_ts[(i, j, t)] = ss_val + LT_DEMAND.get((i, j)
                    , 0.0)

                try:
                    qty = Qv[i, j, t].X
                except (gp.GurobiError, KeyError):
                    qty = 0.0
                if qty > 1e-6:
                    Q_executed[(i, j, t)] = qty
                    arr = t + lead_time_for(j)
                    if arr <= N_PERIODS:
                        pipeline_stock[(i, j, arr)] =
                            pipeline_stock.get((i, j, arr), 0.0) +
                            qty

                if j in DEALERS:
                    try:
                        qty_do = Q_DOv[i, j, t].X
                    except (gp.GurobiError, KeyError):
                        qty_do = 0.0
                    if qty_do > 1e-6:
                        Q_DO_executed[(i, j, t)] = qty_do
                        arr = t + lead_time_for(j, emergency=True
                            )
                        if arr <= N_PERIODS:
                            pipeline_do[(i, j, arr)] =
                                pipeline_do.get((i, j, arr), 0.0) +
                                qty_do

    if verbose:
        print(f"\n[{objective_mode.upper()}] Rolling horizon
            complete.\n")
```

```
    return dict(inv_value_ts=inv_value_ts, SS_ts=SS_ts, EOQ_ts=EOQ_ts,
                ROP_ts=ROP_ts, cost_ts=cost_ts, ls_ts=ls_ts,
                mip_gap_ts=mip_gap_ts, solve_time_ts=solve_time_ts,
                objective_mode=objective_mode)


def print_results(results_bo, results_cost):
    sep = "=" * 80
    mid = "-" * 80
    print(f"\n{sep}\n   OBJECTIVE VALUES\n{sep}")
    print(f"   {'Period':<8}   {'OBJ1 (BO units)':>22}   {'OBJ2 (cost SEK)':>22}")
    print(mid)
    for t in PERIODS:
        print(f"   M{t:<7}   {results_bo['cost_ts'].get(t, float('nan')):>22.4f}"
              f"   {results_cost['cost_ts'].get(t, float('nan')):>22.2f}")
    print(sep)

    for label, key in [("SAFETY STOCK", "SS_ts"), ("ROP", "ROP_ts"), ("EOQ", "EOQ_ts")]:
        print(f"\n{sep}\n   {label} at M{N_PERIODS}\n{sep}")
        for i in ITEMS:
            for j in LOCATIONS:
                v1 = results_bo[key].get((i, j, N_PERIODS), 0.0)
                v2 = results_cost[key].get((i, j, N_PERIODS), 0.0)
                print(f"   {i:<10}   {j:<6}   OBJ1={v1:>8.2f}   OBJ2={v2:>8.2f}")
        print(sep)

    ls_ts = results_cost.get("ls_ts", {})
    if any(ls_ts.get((i, t), 0) > 0 for i in ITEMS for t in PERIODS):
        print(f"\n{sep}\n   LOST SALES      OBJ2\n{sep}")
        for i in ITEMS:
            for t in PERIODS:
                ls = ls_ts.get((i, t), 0.0)
                print(f"   {i:<10}   M{t}   lost={ls:.4f}   cost={ls*PRICE[i]*C_LOST_SALES:.2f} SEK")
        print(sep)


# ============================================================
# ORIGINAL PLOT
# ============================================================

def plot_results(results_bo: dict, results_cost: dict,
```

```
                save_path: str = "inventory_results.png") -> str
                    :
    import matplotlib
    matplotlib.use("Agg")
    import matplotlib.pyplot as plt
    import matplotlib.gridspec as gridspec
    import numpy as np

    plt.rcParams.update({
        "font.family":        "DejaVu Sans",
        "font.size":          9,
        "axes.titlesize":     10,
        "axes.titleweight":   "bold",
        "axes.labelsize":     8.5,
        "axes.spines.top":    False,
        "axes.spines.right":  False,
        "axes.grid":          True,
        "grid.color":         "#e2e8f0",
        "grid.linewidth":     0.6,
        "lines.linewidth":    2.0,
        "lines.markersize":   5.5,
        "xtick.direction":    "out",
        "ytick.direction":    "out",
        "figure.facecolor":   "#f8fafc",
        "axes.facecolor":     "#ffffff",
    })

    C = {
        "PartA":  "#2563eb",
        "PartB":  "#dc2626",
        "CDC":    "#16a34a",
        "SDC":    "#d97706",
        "D1":     "#7c3aed",
        "demand": "#94a3b8",
        "SS":     "#16a34a",
        "ROP":    "#d97706",
        "EOQ":    "#0891b2",
        "OBJ_BO": "#dc2626",
        "OBJ_CO": "#2563eb",
    }

    T       = PERIODS
    x       = np.array(T)
    xlabels = [f"M{t}" for t in T]
    n_items = len(ITEMS)

    fig = plt.figure(figsize=(16, 21), constrained_layout=False)
    fig.patch.set_facecolor("#f8fafc")
    fig.suptitle(
        "Rolling-Horizon Inventory Optimisation   |   Academic
          Model\n"
```

```
        "Network: CDC \u2192 SDC \u2192 D1      "
        "Items: PartA, PartB      Horizon: 5 periods",
        fontsize=12, fontweight="bold", color="#1e293b", y=0.985,
    )

    gs = gridspec.GridSpec(
        5, 1, figure=fig,
        top=0.955, bottom=0.04, hspace=0.65,
        height_ratios=[1.0, 1.5, 1.1, 1.5, 0.9],
    )

    def _fmt_x(ax, xs=x, labs=xlabels):
        ax.set_xticks(xs)
        ax.set_xticklabels(labs)
        ax.set_xlim(xs[0] - 0.5, xs[-1] + 0.5)

    ax0  = fig.add_subplot(gs[0])
    ax0r = ax0.twinx()
    bo_vals   = [results_bo["cost_ts"].get(t, float("nan"))   for
        t in T]
    cost_vals = [results_cost["cost_ts"].get(t, float("nan")) for
        t in T]
    h_bo, = ax0.plot(x,  bo_vals,   "o-",  color=C["OBJ_BO"],
       label="OBJ1 \u2013 Back-order units")
    h_co, = ax0r.plot(x, cost_vals, "s--", color=C["OBJ_CO"],
       label="OBJ2 \u2013 Total cost (SEK)")
    ax0.set_ylabel("Back-order units",  color=C["OBJ_BO"],
       labelpad=6)
    ax0r.set_ylabel("Total cost (SEK)", color=C["OBJ_CO"],
       labelpad=6)
    ax0.tick_params(axis="y", colors=C["OBJ_BO"])
    ax0r.tick_params(axis="y", colors=C["OBJ_CO"])
    _fmt_x(ax0)
    ax0.set_title("Objective Values per Period")
    ax0.legend(handles=[h_bo, h_co], loc="upper right", fontsize
       =8, framealpha=0.85)
    ax0r.grid(False)
    ax0r.spines["top"].set_visible(False)

    gs1 = gridspec.GridSpecFromSubplotSpec(1, n_items,
       subplot_spec=gs[1], wspace=0.38)
    for ci, item in enumerate(ITEMS):
        ax = fig.add_subplot(gs1[ci])
        for loc in LOCATIONS:
            vals = [results_cost["inv_value_ts"].get((item, loc,
               t), 0.0) / PRICE[item]
                    for t in T]
            ax.plot(x, vals, "o-", color=C[loc], label=loc)
        dem = [D_ACTUAL.get((item, "D1", t), 0.0) for t in T]
        ax.bar(x, dem, width=0.38, alpha=0.18, color=C["demand"],
            label="D1 demand", zorder=1)
```

```
        _fmt_x(ax)
        ax.set_title(f"Inventory on Hand \u2014 {item}")
        ax.set_ylabel("Units")
        ax.legend(fontsize=7, loc="upper right", framealpha=0.85)

    ax2 = fig.add_subplot(gs[2])
    combo_labs = [f"{i}\n{j}" for i in ITEMS for j in LOCATIONS]
    x2  = np.arange(len(combo_labs))
    w   = 0.26
    ss_v  = [results_cost["SS_ts"].get( (i, j, N_PERIODS), 0.0)
       for i in ITEMS for j in LOCATIONS]
    rop_v = [results_cost["ROP_ts"].get((i, j, N_PERIODS), 0.0)
       for i in ITEMS for j in LOCATIONS]
    eoq_v = [results_cost["EOQ_ts"].get((i, j, N_PERIODS), 0.0)
       for i in ITEMS for j in LOCATIONS]
    ax2.bar(x2 - w, ss_v,  width=w, color=C["SS"],  label="Safety
       Stock (SS)",   alpha=0.88, zorder=3)
    ax2.bar(x2,     rop_v, width=w, color=C["ROP"], label="
       Reorder Point (ROP)", alpha=0.88, zorder=3)
    ax2.bar(x2 + w, eoq_v, width=w, color=C["EOQ"], label="Order
       Qty (EOQ)",     alpha=0.88, zorder=3)
    ax2.set_xticks(x2)
    ax2.set_xticklabels(combo_labs, fontsize=7.5)
    ax2.set_title(f"Safety Stock / ROP / Order Qty at Period M{
       N_PERIODS}  \u2014  OBJ2 (Cost Min)")
    ax2.set_ylabel("Units")
    ax2.legend(fontsize=7.5, loc="upper right", framealpha=0.85)

    gs3 = gridspec.GridSpecFromSubplotSpec(1, n_items,
       subplot_spec=gs[3], wspace=0.38)
    x3  = np.arange(len(T))
    w3  = 0.26
    for ci, item in enumerate(ITEMS):
        ax = fig.add_subplot(gs3[ci])
        for li, loc in enumerate(LOCATIONS):
            vals = [results_cost["EOQ_ts"].get((item, loc, t),
               0.0) for t in T]
            ax.bar(x3 + (li - 1) * w3, vals, width=w3, color=C[
               loc], label=loc, alpha=0.88, zorder=3)
        ax.set_xticks(x3)
        ax.set_xticklabels(xlabels)
        ax.set_xlim(-0.6, len(T) - 0.4)
        ax.set_title(f"Order Quantities per Period \u2014 {item}"
           )
        ax.set_ylabel("Units ordered")
        ax.legend(fontsize=7.5, framealpha=0.85)

    ax4 = fig.add_subplot(gs[4])
    ls_data = results_cost.get("ls_ts", {})
    any_ls  = False
    for item in ITEMS:
```

```
        ls_vals = [ls_data.get((item, t), 0.0) for t in T]
        ax4.plot(x, ls_vals, "o-", color=C[item], label=item)
        if any(v > 0 for v in ls_vals):
            any_ls = True
    ax4.axhline(0, color="#cbd5e1", linewidth=0.9, linestyle="--")
    _fmt_x(ax4)
    ax4.set_title("Lost Sales per Period  \u2014  OBJ2 (Cost Min)")
    ax4.set_ylabel("Units lost")
    ax4.legend(fontsize=8, framealpha=0.85)
    if not any_ls:
        ax4.text(0.5, 0.55, "No lost sales recorded",
                 transform=ax4.transAxes, ha="center", va="center",
                 fontsize=9, color="#64748b", style="italic")

    fig.savefig(save_path, dpi=150, bbox_inches="tight",
                facecolor=fig.get_facecolor())
    plt.close(fig)
    print(f"\n[PLOT] Figure saved -> {save_path}")
    return save_path


# ============================================================
# ANALYTICAL PLOTS     three numerical-analysis diagnostics
# ============================================================

def run_bigM_sensitivity(scales=(0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 5.0)):

    print("\n[SENSITIVITY] Big-M scale sweep:", scales)
    records = []
    for s in scales:
        mb = _compute_mbig(scale=s)
        try:
            r = rolling_horizon_solve(objective_mode=OBJ_COST,
                                      m_big_override=mb, verbose=False)
            total_cost  = sum(r["cost_ts"].values())
            total_time  = sum(r["solve_time_ts"].values())
            avg_gap     = sum(r["mip_gap_ts"].values()) / len(PERIODS)
            records.append((s, total_cost, total_time, avg_gap, True))
            print(f"  scale={s:.2f}  cost={total_cost:.2f}  time={total_time:.3f}s  gap={avg_gap:.4f}%")
        except RuntimeError as e:
            print(f"  scale={s:.2f}  INFEASIBLE: {e}")
            records.append((s, float("nan"), float("nan"), float("nan"), False))
```

```
    return records


def run_horizon_sensitivity(window_sizes=(1, 2, 3, 4, 5)):

    print("\n[SENSITIVITY] Horizon length sweep:", window_sizes)
    records = []
    for W in window_sizes:
        try:
            r = rolling_horizon_solve(objective_mode=OBJ_COST,
                                      fixed_horizon=W, verbose=
                                         False)
            total_cost = sum(r["cost_ts"].values())
            total_bo   = sum(
                r.get("mip_gap_ts", {t: 0 for t in PERIODS}).
                   values())  # placeholder
            avg_gap    = sum(r["mip_gap_ts"].values()) / len(
               PERIODS)
            total_time = sum(r["solve_time_ts"].values())
            records.append((W, total_cost, avg_gap, total_time))
            print(f"  W={W}  cost={total_cost:.2f}  gap={avg_gap
               :.4f}%  time={total_time:.3f}s")
        except RuntimeError as e:
            print(f"  W={W}  FAILED: {e}")
            records.append((W, float("nan"), float("nan"), float(
               "nan")))
    return records


def plot_analytical(results_cost: dict,
                    bigm_records,
                    horizon_records,
                    save_path: str = "analytical_results.png") ->
                        str:

    import matplotlib
    matplotlib.use("Agg")
    import matplotlib.pyplot as plt
    import matplotlib.gridspec as gridspec
    import numpy as np

    plt.rcParams.update({
        "font.family":       "DejaVu Sans",
        "font.size":          9,
        "axes.titlesize":    10,
        "axes.titleweight": "bold",
        "axes.labelsize":     8.5,
        "axes.spines.top":   False,
        "axes.spines.right":False,
        "axes.grid":         True,
        "grid.color":        "#e2e8f0",
```

```
        "grid.linewidth":   0.6,
        "lines.linewidth":  2.0,
        "lines.markersize": 6.0,
        "figure.facecolor": "#f8fafc",
        "axes.facecolor":   "#ffffff",
    })

    # Colour tokens
    C_COST  = "#2563eb"   # blue
    C_GAP   = "#dc2626"   # red
    C_TIME  = "#16a34a"   # green
    C_COST2 = "#7c3aed"   # violet (second cost series)
    ALPHA_BAND = 0.13

    T       = PERIODS
    x       = np.array(T)
    xlabels = [f"M{t}" for t in T]

    fig = plt.figure(figsize=(16, 18), constrained_layout=False)
    fig.patch.set_facecolor("#f8fafc")
    fig.suptitle(
        "Numerical␣Analysis␣of␣the␣Rolling-Horizon␣MIP\n"
        "Big-M␣Sensitivity␣␣  ␣␣Horizon␣Myopia␣␣  ␣␣MIP␣Gap␣
           Convergence",
        fontsize=13, fontweight="bold", color="#1e293b", y=0.985,
    )

    gs = gridspec.GridSpec(
        3, 1, figure=fig,
        top=0.95, bottom=0.06, hspace=0.52,
        height_ratios=[1.0, 1.0, 1.0],
    )

    #         Panel A : MIP Gap Convergence


    axA  = fig.add_subplot(gs[0])
    axAr = axA.twinx()

    gap_vals  = [results_cost["mip_gap_ts"].get(t, float("nan"))
       for t in T]
    time_vals = [results_cost["solve_time_ts"].get(t, float("nan"
       )) for t in T]

    hA1, = axA.plot(x,  gap_vals,  "o-",  color=C_GAP,  label="
       MIP␣gap␣(%)")
    hA2, = axAr.plot(x, time_vals, "s--", color=C_TIME, label="
       Solve␣time␣(s)")

    axA.fill_between(x, 0, gap_vals, alpha=ALPHA_BAND, color=
       C_GAP)
```

```
    axA.set_ylabel("MIP␣optimality␣gap␣(%)", color=C_GAP,
       labelpad=6)
    axAr.set_ylabel("Wall-clock␣solve␣time␣(s)", color=C_TIME,
       labelpad=6)
    axA.tick_params(axis="y", colors=C_GAP)
    axAr.tick_params(axis="y", colors=C_TIME)
    axA.set_xticks(x); axA.set_xticklabels(xlabels)
    axA.set_xlim(x[0] - 0.4, x[-1] + 0.4)
    axA.set_title(
        "Panel␣A␣   ␣MIP␣Gap␣Convergence␣Across␣Rolling-Horizon␣
           Periods\n"
        r"$\bf{Numerical\,analysis\,context:}$␣"
        "As␣the␣remaining␣horizon␣shrinks,␣the␣feasible␣region␣
           contracts␣"
        "   ␣branch-and-bound␣closes␣the␣gap␣faster.",
        loc="left", fontsize=8.5, color="#334155",
    )
    axA.legend(handles=[hA1, hA2], loc="upper␣right", fontsize=8,
        framealpha=0.85)
    axAr.grid(False)
    axAr.spines["top"].set_visible(False)

    # Annotate each point with its gap value
    for xi, (g, s) in zip(x, zip(gap_vals, time_vals)):
        if not math.isnan(g):
            axA.annotate(f"{g:.3f}%", xy=(xi, g),
                         xytext=(0, 7), textcoords="offset␣points
                            ",
                         ha="center", fontsize=7.5, color=C_GAP)

    #         Panel B : Big-M Sensitivity


    axB  = fig.add_subplot(gs[1])
    axBr = axB.twinx()

    bm_scales    = [r[0] for r in bigm_records]
    bm_costs     = [r[1] for r in bigm_records]
    bm_gaps      = [r[3] for r in bigm_records]
    bm_feasible  = [r[4] for r in bigm_records]

    # Mask infeasible
    bm_costs_plot = [c if f else float("nan") for c, f in zip(
       bm_costs, bm_feasible)]
    bm_gaps_plot  = [g if f else float("nan") for g, f in zip(
       bm_gaps,  bm_feasible)]

    hB1, = axB.plot(bm_scales, bm_costs_plot, "o-",  color=C_COST
       ,  label="Total␣cost␣(SEK)")
    hB2, = axBr.plot(bm_scales, bm_gaps_plot, "s--", color=C_GAP,
          label="Avg␣MIP␣gap␣(%)")
```

```
818
819     axB.fill_between(bm_scales, 0, [c if not math.isnan(c) else 0
            for c in bm_costs_plot],
820                      alpha=ALPHA_BAND, color=C_COST)
821
822     # Mark infeasible points
823     for s, f in zip(bm_scales, bm_feasible):
824         if not f:
825             axB.axvline(s, color="#f87171", linewidth=1.2,
                    linestyle=":", alpha=0.7)
826             axB.text(s, axB.get_ylim()[1] * 0.95 if axB.get_ylim
                    ()[1] > 0 else 1,
827                      "INFEAS.", ha="center", fontsize=7, color="#
                         ef4444", rotation=90)
828
829     axB.set_xlabel("Big-M␣scale␣factor␣␣(1.0␣=␣baseline)",
            labelpad=6)
830     axB.set_ylabel("Total␣cost␣over␣horizon␣(SEK)", color=C_COST,
            labelpad=6)
831     axBr.set_ylabel("Average␣MIP␣gap␣(%)", color=C_GAP, labelpad
            =6)
832     axB.tick_params(axis="y", colors=C_COST)
833     axBr.tick_params(axis="y", colors=C_GAP)
834     axB.set_xticks(bm_scales)
835     axB.set_title(
836         "Panel␣B␣   ␣Big-M␣Parameter␣Sensitivity\n"
837         r"$\bf{Numerical\,analysis\,context:}$␣"
838         "Undersized␣M␣causes␣constraint␣tightening␣/␣
                infeasibility;␣"
839         "oversized␣M␣weakens␣the␣LP␣relaxation,␣inflating␣MIP␣
                gaps.",
840         loc="left", fontsize=8.5, color="#334155",
841     )
842     axB.legend(handles=[hB1, hB2], loc="upper␣right", fontsize=8,
            framealpha=0.85)
843     axBr.grid(False)
844     axBr.spines["top"].set_visible(False)
845
846     # Baseline reference line
847     baseline_cost = bm_costs_plot[bm_scales.index(1.0)] if 1.0 in
            bm_scales else None
848     if baseline_cost and not math.isnan(baseline_cost):
849         axB.axhline(baseline_cost, color=C_COST, linewidth=0.8,
850                     linestyle=":", alpha=0.5, label="_nolegend_")
851         axB.text(bm_scales[0], baseline_cost,
852                  f"␣␣baseline\n␣␣{baseline_cost:.0f}␣SEK",
853                  fontsize=7, color=C_COST, va="bottom")
854
855     #         Panel C : Horizon Length vs Solution Quality
```

```
    axC  = fig.add_subplot(gs[2])
    axCr = axC.twinx()

    hz_W     = [r[0] for r in horizon_records]
    hz_costs = [r[1] for r in horizon_records]
    hz_gaps  = [r[2] for r in horizon_records]
    hz_times = [r[3] for r in horizon_records]

    hC1, = axC.plot(hz_W, hz_costs, "o-",  color=C_COST2, label="Total cost (SEK)")
    hC2, = axCr.plot(hz_W, hz_gaps,  "s--", color=C_GAP,   label="Avg MIP gap (%)")
    hC3, = axCr.plot(hz_W, hz_times, "^:", color=C_TIME,  label="Total solve time (s)", alpha=0.8)

    axC.fill_between(hz_W, min([c for c in hz_costs if not math.isnan(c)], default=0),
                     hz_costs, alpha=ALPHA_BAND, color=C_COST2)

    # Annotate cost reduction from W=1 to W=N
    valid_costs = [(w, c) for w, c in zip(hz_W, hz_costs) if not math.isnan(c)]
    if len(valid_costs) >= 2:
        c_min, c_max = valid_costs[0][1], valid_costs[-1][1]
        pct = (c_max - c_min) / c_max * 100 if c_max > 0 else 0
        axC.annotate(
            f" cost \n{pct:+.1f}%",
            xy=(valid_costs[-1][0], valid_costs[-1][1]),
            xytext=(-35, 15), textcoords="offset points",
            arrowprops=dict(arrowstyle="->", color="#475569", lw=1.2),
            fontsize=8, color="#475569",
        )

    axC.set_xlabel("Look-ahead window length  W  (periods)", labelpad=6)
    axC.set_ylabel("Total cost over horizon (SEK)", color=C_COST2, labelpad=6)
    axCr.set_ylabel("Avg MIP gap (%)  /  Total time (s)", color=C_GAP, labelpad=6)
    axC.tick_params(axis="y", colors=C_COST2)
    axCr.tick_params(axis="y", colors=C_GAP)
    axC.set_xticks(hz_W)
    axC.set_xticklabels([f"W={w}" for w in hz_W])
    axC.set_title(
        "Panel C    Rolling-Horizon Myopia: Look-Ahead Window vs. Solution Quality\n"
        r"$\bf{Numerical\,analysis\,context:}$ "
        "W=1 is greedy/myopic; W=N_PERIODS is full-horizon. "
        "Approximation error decays as W increases.",
        loc="left", fontsize=8.5, color="#334155",
```

```
897     )
898     axC.legend(handles=[hC1, hC2, hC3], loc="upper right",
            fontsize=8, framealpha=0.85)
899     axCr.grid(False)
900     axCr.spines["top"].set_visible(False)
901
902     fig.savefig(save_path, dpi=150, bbox_inches="tight",
            facecolor=fig.get_facecolor())
903     plt.close(fig)
904     print(f"\n[PLOT] Analytical figure saved -> {save_path}")
905     return save_path
906
907
908 # ============================================================
909 # ENTRY POINT
910 # ============================================================
911
912 if __name__ == "__main__":
913     print("\n  INPUT DATA SUMMARY")
914     for i in ITEMS:
915         print(f"  {i}: price={PRICE[i]} SEK  weight={WEIGHT[i]}g
              vol={VOLUME[i]} m3")
916
917     results_bo   = rolling_horizon_solve(objective_mode=
            OBJ_BACKORDER)
918     results_cost = rolling_horizon_solve(objective_mode=OBJ_COST)
919
920     print_results(results_bo, results_cost)
921     plot_results(results_bo, results_cost, save_path="
            inventory_results.png")
922
923     #         Analytical experiments
924     print("\n" + "=" * 65)
925     print("  RUNNING ANALYTICAL SENSITIVITY EXPERIMENTS")
926     print("=" * 65)
927
928     bigm_records    = run_bigM_sensitivity(scales=[0.5, 0.75,
            1.0, 1.5, 2.0, 3.0, 5.0])
929     horizon_records = run_horizon_sensitivity(window_sizes=[1, 2,
            3, 4, 5])
930
931     plot_analytical(
932         results_cost   = results_cost,
933         bigm_records   = bigm_records,
934         horizon_records= horizon_records,
935         save_path      = "analytical_results.png",
936     )
937
938     print("\nDone.")
```

Bachelor's Theses in Mathematical Sciences 2026:K20
ISSN 1654-6229

LUNFNA-4072-2026

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/