Development of Vehicle Routing Problem (VRP) Models in Python to Optimize Last-Mile Distribution Routes in an SME

Maya Bautista Jukari[1]*, Navarro Ordoñez Diana Mariel[1], Rivera Escalante Daniel[1], Meneses Sánchez Valeria[1], Acosta Martínez Nayeli Abigail[1]

Industrial Engineering, Instituto Tecnológico de Pachuca, Pachuca de Soto, Mexico.

DOI: https://doi.org/10.47772/IJRISS.2026.102500010

Received: 21 October 2026; Accepted: 26 October 2026; Published: 14 November 2026

ABSTRACT

Small and medium-sized enterprises (SMEs) that produce perishable food often plan last-mile distribution reactively, which raises fuel consumption, extends working days and forces vehicles to return to the plant for additional product. This study develops and evaluates vehicle routing problem (VRP) models implemented in Python for a family-owned cheese SME located in Apaxco de Ocampo, State of Mexico, which serves about 35 commercial destinations per day with more than 900 kg of product using two light trucks of 600 and 800 kg. The research followed the Design Thinking approach in five phases. The empathize and define phases characterized the current operation, three delivery corridors and the isolated fuel cost of each destination. The ideate phase formulated a heterogeneous capacitated VRP with working-day limits, multiple trips and order release times. The prototype phase implemented the model in Python using a Clarke-Wright savings heuristic, a simulated annealing metaheuristic and an exact mixed-integer verification solved with HiGHS. In the test phase, three scenarios were compared under identical orders, vehicles and assumptions: the current dispatching rules, a static scenario with all orders ready at 07:00 and a dynamic scenario with staggered production releases. Under the dynamic scenario, the model reduced the travelled distance from 382.0 to 285.4 km (25.3 %), daily fuel cost from MXN 1,293 to MXN 966, trips from six to four and minutes beyond the working day by 61.2 %. When production and dispatch were synchronized, distance fell to 152.8 km (60.0 %) with two trips and no overtime. A Monte Carlo experiment with 30 simulated days yielded a mean distance reduction of 35.1 %, and the exact verification confirmed the optimal sequence of every route. The results show that route optimization must be accompanied by the synchronization of production and dispatch.

Keywords: Vehicle Routing Problem, Last-Mile Distribution, Python, Design Thinking, Small And Medium-Sized Enterprises

INTRODUCTION

Last-mile distribution is the most expensive and least efficient segment of the supply chain, because it concentrates small, scattered deliveries that are subject to time, access and capacity constraints. Consequently, operations research has developed multiple variants of the vehicle routing problem (VRP), originally formulated by Dantzig and Ramser (1959), to decide which customers each vehicle serves and in what order. Recently, these variants have been extended to e-commerce settings, delivery options, relay points and alternative vehicles (Dribel et al., 2025; Yang et al., 2025; Zhou et al., 2018); however, much of this evidence comes from large-scale operators, with technological infrastructure and data volumes that do not match the reality of small and medium-sized enterprises (SMEs).

In the case of perishable food, the problem becomes more complex. On the one hand, product quality decreases with transport time and storage conditions; and on the other, the systematic review by Utama et al. (2020) shows that minimizing total distribution cost is the most widely used objective and that metaheuristics are the predominant solution method. In the distribution of dairy products in particular, Sanabria-Rey et al. (2021) showed that a simple tool, based on a biased-randomization heuristic, can be competitive in working-day duration, number of vehicles and occupancy, which suggests that SMEs can benefit from accessible models if these are adapted to their actual operation.

This study focuses on a family-owned SME in the dairy sector, a cheese producer located in Apaxco de Ocampo, State of Mexico, whose name is withheld for confidentiality. According to the information provided by the company, it produces and distributes on the same day, without a separate finished-goods warehouse; it serves customers in Apaxco, Tequixquiac, Zumpango and the Metropolitan Area of the State of Mexico; it operates two light trucks with nominal capacities of 600 and 800 kg; and it reports approximately 35 shipments per day that together exceed 900 kg. Likewise, the company identifies as its main problems the high distribution costs, fuel consumption, route duration and returns to the plant to load more product, situations that stem from a reactive assignment of orders as goods leave production.

Although the literature offers formulations for the capacitated VRP (CVRP), the VRP with time windows (VRPTW) and the multi-trip VRP (Pan et al., 2021; González-Romero et al., 2026), as well as open-source Python libraries (Errami et al., 2024; Klein and Schiffer, 2024), an application gap persists: for an SME that produces perishable food and manufactures and dispatches simultaneously, no model has been documented that integrates fleet heterogeneity, the working-day limit, replenishment trips and order release times, developed with open tools and guided by a design process centered on the people who operate the distribution. Therefore, this study adopts the Design Thinking methodology (Brown, 2008; Hasso Plattner Institute of Design at Stanford, 2010) as a framework to move from understanding the problem to evaluating a computational prototype.

Accordingly, the general objective of this research is to develop and evaluate vehicle routing models implemented in Python to optimize the last-mile distribution routes of the SME under study. Specifically, it seeks to: (a) characterize the current operation and the distribution corridors based on the company's information; (b) formulate a capacitated, heterogeneous, multi-trip VRP model with order release times; (c) implement the model in Python through a constructive heuristic, a metaheuristic and an exact verification; and (d) compare the current operation with optimized scenarios under the same orders, vehicles and assumptions, as well as analyze the sensitivity of the results.

Based on the above, the hypothesis is that route planning through a multi-trip VRP model with order release reduces the distance traveled, fuel cost and return trips to the plant compared with the current reactive dispatching rules, although the magnitude of the reduction depends on the synchronization between production and dispatch. Therefore, the contribution of the study is twofold: on the practical level, it offers the company a reproducible, low-cost tool to plan its routes; on the methodological level, it shows how Design Thinking can structure the development of optimization models in SMEs, so that modeling decisions respond to verifiable operational needs.

LITERATURE REVIEW

The literature on last-mile routing can be organized into four thematic axes. The first brings together the structural VRP variants that emerged from e-commerce, including two-echelon systems with delivery options (Yu, Lin, et al., 2024; Zhou et al., 2018), schemes with relay points, crowdshipping and roaming locations (Dribel et al., 2025; Liao et al., 2025; Yang et al., 2025) and models with front warehouses or delivery robots (Li and Yang, 2024; Malik et al., 2025). Although these contributions broaden the scope of the problem, they depend on infrastructure that a producing SME does not have, so their contribution to this study lies mainly in the solution methods they use.

The second axis corresponds to the distribution of perishable products and food. Hsiao et al. (2018) integrated quality deterioration into cold chains, while SteadieSeifi et al. (2017) addressed quality preservation in multimodal networks. Similarly, Thipparthy et al. (2024) optimized food deliveries with several items per order, and Sanabria-Rey et al. (2021) solved a dairy case with a spreadsheet tool. Overall, the review by Utama et al. (2020) confirms that these problems are mostly solved with metaheuristics that minimize total distribution cost.

The third axis groups dynamic and multi-trip models, which are closer to the operation of a company that produces and dispatches during the same working day. Pan et al. (2021) formulated a multi-trip VRP with

time-dependent travel times and loading time at the depot; González-Romero et al. (2026) incorporated economic, environmental and social criteria into a multi-trip model; and Maghfiroh and Hanaoka (2018) showed, albeit in a humanitarian context, the advantages of a heterogeneous fleet with multiple trips under uncertain demand, a topic that Ortuño et al. (2021) place within last-mile humanitarian logistics. In turn, Fonseca-Galindo et al. (2022), Dahlan et al. (2025) and Rashidi et al. (2025) used trajectory, traffic or historical route data to adapt routes to changing conditions, while Álvarez-Miranda et al. (2025) addressed the prior tactical decision of dividing the territory into delivery districts. Likewise, Yu, Li, et al. (2024) incorporated the cost of emissions into last-mile routing.

The fourth axis refers to applicability in SMEs and to implementation tools. Nieto de Almeida et al. (2012) developed a low-cost constructive heuristic for distribution SMEs, and He et al. (2018) solved a VRP with a working-day limit for SME carriers. Regarding tools, Errami et al. (2024) and Klein and Schiffer (2024) published open-source Python libraries that facilitate the exact or heuristic solution of the VRP, while the OR-Tools documentation describes the modeling of heterogeneous capacities and time windows (Google for Developers, n.d.-a, n.d.-b, n.d.-c). Table 1 summarizes the 25 studies consulted, and Table 2 contrasts the modeling attributes they address with those required by the case study.

To integrate the corpus, the following question was defined: which VRP models and solution methods have been proposed for last-mile distribution, and which attributes of capacity, time, multiple trips, dynamism, perishability and application in SMEs do they incorporate? Peer-reviewed articles, chapters and proceedings with a DOI, published between 2012 and 2026, that proposed or applied a last-mile routing model or a tool to solve it were considered. For each study, the year, authors, objective, main limitation with respect to the case study and most relevant results were recorded, following a study-by-study synthesis structure.

Table 1. Synthesis of the studies included in the literature review

Year	Author	Objective	Limitation	Results
2012	Nieto de Almeida et al.	Design an efficient, low-cost constructive heuristic for the VRPTW of distribution SMEs.	Validated with Solomon benchmark instances rather than with company data.	Highly competitive performance as a constructive method, executable on mid- or low-range computers.
2017	SteadieSeifi et al.	Plan multimodal flows of perishable products with repositioning of reusable containers.	Long-haul transport network approach, not last-mile routing.	Multicommodity flow MIP model and an ALNS algorithm with new neighborhoods, compared with a commercial solver.
2018	He et al.	Solve a rich VRP with time windows and a working-day limit for SME carriers.	Specific case in Changzhou; service times differentiated by customer type are assumed.	The hybrid ant colony–tabu search algorithm showed that collaborative planning increases profits.
2018	Hsiao et al.	Plan last-mile distribution in fruit and vegetable cold chains considering their quality.	Does not consider multiple trips per vehicle or staggered order release.	A genetic algorithm determined fleet, sequence and shipments, meeting different quality levels at a controlled cost.
2018	Maghfiroh and Hanaoka	Solve a dynamic truck and trailer VRP with a heterogeneous fleet and multiple trips in disaster response.	Humanitarian context and land transport only.	Simulated annealing with variable neighborhood search showed the benefits of flexible routing under uncertain demand.
2018	Zhou et al.	Minimize the cost of a multi-depot two-echelon	Requires satellites and pickup points, unrelated to	A hybrid multi-population genetic algorithm proved

		VRP with delivery options in e-commerce.	an SME with a single depot.	effective on instance families and on a real case.
2020	Utama et al.	Systematically review the VRP for perishable goods (59 articles, 2001–2020).	Documentary review; does not implement models.	Metaheuristics and the minimization of total distribution cost as the objective function predominate.
2021	Ortuño et al.	Describe the optimization of humanitarian logistics, including last-mile distribution.	Conceptual encyclopedia chapter, without an applied case.	Systematizes evacuation, location and last-mile distribution problems and their optimization approaches.
2021	Pan et al.	Minimize distance in a multi-trip VRP with time-dependent travel times, time windows and loading at the depot.	Evaluated on test instances; does not consider product availability.	A hybrid ALNS–VND proved robust to different speed profiles and trip duration limits.
2021	Sanabria-Rey et al.	Solve last-mile dairy deliveries as a VRPTW minimizing the total distribution duration.	Spreadsheet tool applied to a single case (Bogotá).	The multi-start algorithm with biased randomization was competitive in duration, vehicles, occupancy and costs.
2022	Fonseca-Galindo et al.	Solve a dynamic CVRP with stochastic customers through a multi-agent system and trajectory data mining.	Slightly lower performance than the static VRP; oriented to large parcel volumes.	Generated dynamic routes with operational gains by eliminating storage and batching stages.
2024	Errami et al.	Develop VRPSolverEasy, a Python library to solve a rich VRP exactly.	Requires adapting the case to the library's structure; does not present a business application.	Allows modeling depots, customers, links and vehicle types without programming MIP models.
2024	Klein and Schiffer	Develop RoutingBlocks, an open-source Python package for VRPs with intermediate stops.	Algorithm development tool, without a business case.	Modular C++ components exposed in Python that separate the algorithm from the problem configuration.
2024	Li and Yang	Solve a last-mile VRP with handling cost in the front-warehouse mode.	Assumes front warehouses and split deliveries, different from simultaneous production and dispatch.	Pickup-and-delivery formulation with handling costs solved through variable neighborhood search.
2024	Thipparthy et al.	Optimize food delivery routes with several items per order and time windows.	The authors present it as a preliminary exploration of the problem.	A genetic algorithm minimized the fulfillment cost; the cases confirmed its feasibility.
2024	Yu, Li, et al.	Solve a low-emission last-mile problem through a discrete marine predators algorithm.	Focused on light urban electric vehicles; does not consider perishable products.	Integrates carbon emissions into the routing cost and uses neighborhood search to improve routes.
2024	Yu, Lin, et al.	Design a fast simulated annealing for a multi-depot two-echelon VRP	Does not report limitations; the two-echelon structure does not	Found 30 of the 36 best-known solutions with superior computational

		with delivery options.	correspond to an SME.	performance.
2025	Álvarez-Miranda et al.	Design delivery districts with a multi-criteria approach and a lexicographic compactness metric (Chilean postal service).	Tactical zoning decision, not daily route sequencing.	Compact and balanced districts useful for assigning delivery territories.
2025	Dahlan et al.	Propose an adaptive last-mile routing framework with crowdsourced data in dynamic traffic.	Depends on the availability of real-time crowdsourced data.	Adjusts routes to traffic disruptions based on user-generated information.
2025	Dribel et al.	Optimize the last mile with relay points and crowdshipping through tabu search.	Relay scheme oriented to end consumers; does not report explicit limitations.	Showed potential to improve operational efficiency in several scenarios.
2025	Liao et al.	Formulate a VRP with roaming locations for simultaneous pickup and delivery services.	Roaming locations are specific to deliveries to end consumers.	Proposes a mixed-integer linear formulation and a large neighborhood search.
2025	Malik et al.	Coordinate collaborative parcel delivery with drones and quadruped robots (CODEL).	Emerging technologies beyond the operational reach of an SME.	Mixed-integer linear formulation with energy consumption constraints for robot coordination.
2025	Rashidi et al.	Generate practical last-mile routes with an insertion heuristic informed by historical data.	Trade-off between driver preferences and travel time; requires historical route data.	Produced 22.4 % and 24.1 % more high-quality solutions, with 20.6 % and 13.9 % more median travel time.
2025	Yang et al.	Solve the VRP with delivery options through a neighborhood-based matheuristic.	Delivery options (lockers and pickup points) do not apply to supplying retail stores.	Combines neighborhood search with mathematical programming to improve solutions.
2026	González-Romero et al.	Solve a multi-trip VRP that balances economic, environmental and social sustainability.	Validated on a single case study.	Overall improvement of 43 % (41 % economic, 29 % environmental and 61 % social) compared with the initial situation.

Note: Studies sorted by year of publication. The limitation is expressed with respect to the case study when the work does not state it explicitly.

Source: Own elaboration based on the included studies.

As shown in Table 1, the studies with real-world application are concentrated on urban parcel operators or on companies that have their orders available before starting deliveries. Building on this, Table 2 shows that no work in the corpus simultaneously combines heterogeneous capacity, a working-day limit, multiple trips, order release during the shift, perishable products, an SME context and an open Python implementation, a combination that defines the gap addressed by this study.

Table 2. Modeling attributes addressed by the studies with the application closest to the case

Study	Capacity	Working day or windows	Multiple trips	Dynamic or release	Perishables	SME	Open Python
Nieto de Almeida et	●	●	–	–	–	●	–

al. (2012)							
SteadieSeifi et al. (2017)	●	–	–	–	●	–	–
He et al. (2018)	●	●	–	●	–	●	–
Hsiao et al. (2018)	●	●	–	–	●	–	–
Maghfiroh and Hanaoka (2018)	●	–	●	●	–	–	–
Zhou et al. (2018)	●	–	–	–	–	–	–
Utama et al. (2020)	●	–	–	–	●	–	–
Pan et al. (2021)	●	●	●	●	–	–	–
Sanabria-Rey et al. (2021)	●	●	–	–	●	–	–
Fonseca-Galindo et al. (2022)	●	–	–	●	–	–	–
Errami et al. (2024)	●	●	–	–	–	–	●
Klein and Schiffer (2024)	●	–	●	–	–	–	●
Thipparthy et al. (2024)	●	●	–	–	●	–	–
Dahlan et al. (2025)	–	–	–	●	–	–	–
González-Romero et al. (2026)	●	●	●	–	–	–	–
This study	●	●	●	●	●	●	●

Note: ● = attribute explicitly addressed; – = not addressed. Studies whose attributes correspond to e-commerce schemes unrelated to the case are excluded.

Source: Own elaboration based on Table 1.

THEORETICAL FRAMEWORK

The VRP consists of designing a set of minimum-cost routes that start and end at a depot, so that each customer is visited exactly once and the operational constraints are met (Dantzig and Ramser, 1959). In its capacitated version (CVRP), the sum of the demands assigned to a vehicle cannot exceed its capacity, and when the fleet is heterogeneous each vehicle has a different limit (Google for Developers, n.d.-a). Adding time windows or a working-day horizon yields the VRPTW (Google for Developers, n.d.-c), and allowing a vehicle to return to the depot to start a new tour yields the multi-trip VRP, in which loading time and the sequence of trips determine temporal feasibility (Pan et al., 2021). Finally, when orders become available during the shift, the release time restricts the moment at which the trip that carries them can begin, a characteristic feature of systems that produce and dispatch simultaneously.

Since these problems are NP-hard, their practical solution combines constructive heuristics and metaheuristics. The Clarke and Wright (1964) savings heuristic merges individual routes according to the distance saved by visiting two customers consecutively, while simulated annealing (Kirkpatrick et al., 1983) explores neighboring solutions and accepts, with decreasing probability, moves that worsen the objective function in order to escape local optima; both techniques have been used successfully in last-mile problems (Maghfiroh and Hanaoka, 2018; Yu, Lin, et al., 2024). In addition, mixed-integer linear programming makes it possible to verify smaller subproblems exactly, such as the optimal sequence of each route, with solvers available in Python scientific libraries (Virtanen et al., 2020).

Lastly, Design Thinking is a human-centered innovation approach that links the understanding of users' needs with technical feasibility and business viability (Brown, 2008). Its process is organized into five iterative phases: empathize, define, ideate, prototype and test (Hasso Plattner Institute of Design at Stanford, 2010). In this study, this approach is used as a methodological framework so that the optimization model is not limited to minimizing a theoretical distance, but responds to the needs of the logistics manager, the drivers and the SME's customers.

METHODOLOGY

Research design

The study adopted an applied design with a quantitative approach and a descriptive-explanatory scope, based on mathematical modeling and computational experimentation. So that the model would respond to the real needs of the operation and not only to a minimum-distance criterion, the research was structured according to the five phases of Design Thinking: empathize, define, ideate, prototype and test (Brown, 2008; Hasso Plattner Institute of Design at Stanford, 2010). As represented in Figure 1, the phases were carried out sequentially, although the results of the test phase fed back into the assumptions, requirements and scenarios, which gave the process an iterative character.

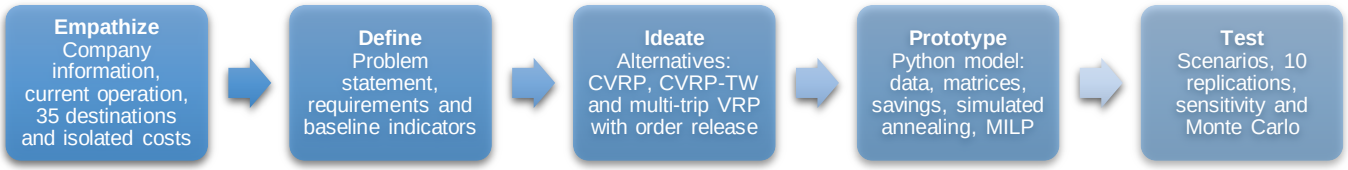


Figure 1. Design Thinking phases applied to the development of the VRP model

Source: Own elaboration, adapted from Brown (2008) and Hasso Plattner Institute of Design at Stanford (2010).

In addition, Table 3 relates each phase to the activities carried out, the products obtained and the criterion used to close the phase. This correspondence makes it possible to trace how each modeling decision derives from a need identified in the company's operation.

Table 3. Integration of the Design Thinking phases with the activities, products and criteria of the study

Phase	Activities	Products	Closing criterion
1. Empathize	Analysis of the company's information; identification of stakeholders (logistics manager, drivers and customers); characterization of 35 potential destinations.	Destination map, corridors, isolated costs and baseline data (Tables 4 to 6; Figures 2 and 3).	Operational problems described with company data.

2. Define	Formulation of the problem statement; translation of needs into requirements; selection of indicators.	"How might we…?" statement, requirements and baseline indicators (Table 7).	Requirements verifiable with a quantitative indicator.
3. Ideate	Comparison of modeling alternatives (CVRP, VRPTW, multi-trip VRP with release).	Selected mathematical formulation (Table 8; Equations 3 to 12).	Model that represents capacity, working day, trips and release.
4. Prototype	Construction of matrices; Python implementation of the heuristic, the metaheuristic, the current rules and the exact verification.	Modular computational prototype (Figure 4; Appendix B).	Reproducible execution and feasible solutions.
5. Test	Comparison of scenarios, replications, MILP verification, sensitivity and Monte Carlo.	Comparative indicators, maps, schedules and robustness analysis (Tables 9 to 14; Figures 5 to 13).	Fulfillment of requirements and iteration of assumptions.

Source: Own elaboration.

Procedure

Phase 1. Empathize: understanding the current operation

The purpose of the first phase was to understand the operation from the perspective of those who carry it out and experience its problems. To this end, the information provided by the company on its history, location, areas served, vehicles, volume, order preparation, working hours and observed problems was analyzed. From this information, three classes of data were distinguished and kept separate to avoid false conclusions: data provided by the company, public data on establishments and engineering assumptions. Table 4 summarizes the reported data and their interpretation for the model.

Table 4. Baseline data reported by the company and their interpretation for the model

Variable	Reported data	Interpretation for the model
Vehicle 1	Nominal capacity of 600 kg	Maximum capacity per trip, subject to validation of payload and volume.
Vehicle 2	Nominal capacity of 800 kg	Maximum capacity per trip, subject to validation of payload and volume.
Aggregate capacity	1,400 kg per simultaneous departure	Nominal sum; it does not imply that 1,400 kg are always ready to load.
Shipments per day	Approximately 35	Interpreted as 35 delivery stops per working day.
Total daily weight	More than 900 kg	Reconciled with larger orders from large-format and wholesale customers.
Typical weight per customer	20–25 kg	35 × 20–25 kg = 700–875 kg; inconsistent with the reported total if all orders fall within that range.
Frequency per zone	Every other day	Allows planning a visit calendar by zone.
Distribution within a zone	Approximately 3 h per zone	Aggregate block per zone, not 3 h per customer.
Delivery hours	07:00–15:00	Horizon H of 480 min.
Return to plant	Vehicles sometimes return for more product	Indicates multiple trips due to late order release or lack of consolidation.

Source: Own elaboration with information provided by the company.

Subsequently, a portfolio of 35 potential commercial destinations (cheese and dairy shops, cheese makers, dairy stores, mini-markets, supermarkets, warehouse clubs and wholesale markets) was compiled from public

directories and business listings (Guiamexican, n.d.; MXFirmas, n.d.). It should be noted that this list is a portfolio for territorial analysis and not a validated customer register; therefore, in this article the destinations are identified only by a number, their business type and their locality (Appendix A). As shown in Figure 2, the destinations were grouped into three corridors: (A) Tequixquiac, with short trips; (B) Zumpango town center and San Juan Zitlaltepec; and (C) the metropolitan corridor toward Tecámac, Ecatepec and Tultitlán.

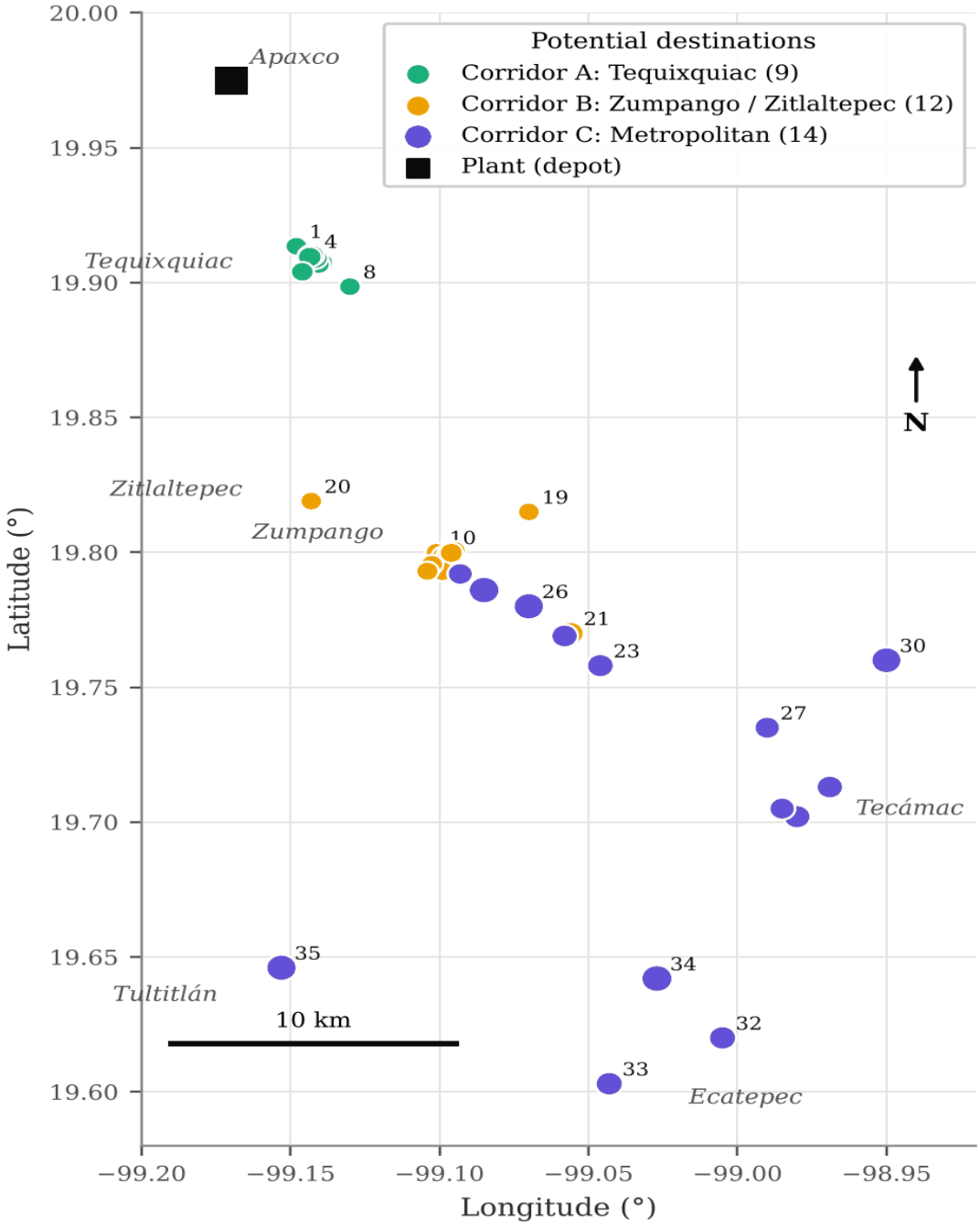


Figure 2. Location of the plant and of the 35 potential destinations by distribution corridor

Note: Marker size is proportional to the assigned demand. Approximate reference coordinates. Source: Own elaboration based on company information, Guiamexican (n.d.) and MXFirmas (n.d.).

To estimate the cost of the operation, a reference fuel economy of 7 km/L was used, a conservative assumption for a light cargo truck in mixed operation (Comisión Nacional para el Uso Eficiente de la Energía [CONUEE], 2025, 2026), together with a regular gasoline price of $23.70 MXN/L, corresponding to the national average reported on October 20, 2026 (GasMapa, 2026). With these values, the liters consumed and the fuel cost of a trip were calculated with Equations 1 and 2, whose parameters are summarized in Table 5.

$$L = d / e \quad (1)$$

$$C = L \times c_f \quad (2)$$

where L is the estimated liters, d the distance traveled in km, e the vehicle's fuel economy in km/L, C the fuel cost in MXN and c_f the price per liter. These calculations do not include idling consumption, variation due to load, slopes or congestion, so they should be replaced by odometer and refueling records when the company has them.

Table 5. Fuel and operating assumptions used in the model

Parameter	Value	Justification or limit
Reference fuel economy, e	7 km/L	Conservative assumption for a light truck with stops and load; not a measured consumption.
Regular gasoline price, c_f	$23.70 MXN/L	National average from October 18 to 20, 2026; the local price may vary.
Cost per kilometer	$3.39 MXN/km	$23.70 ÷ 7 km/L; fuel only, excluding maintenance, tires, depreciation and wages.
Working-day horizon, H	480 min	Stated delivery hours from 07:00 to 15:00.
Loading at the plant per trip, L_c	20 min	Loading and paperwork time before each departure (assumption).
Service per stop, s_i	10 min retail; 15 min large-format or wholesale	Derived from the 3-h block per zone: the 9 stops of corridor A add up to 100 min of service and about 55 min of local travel.
Demand per order, q_i	20–25 kg retail; 28–34 kg large-format; 38–46 kg wholesale	Retail range reported by the company; the larger orders reconcile the daily total above 900 kg.
Order release, r_i	07:00, 09:30 y 11:30	Three production waves with 45 %, 35 % and 20 % assignment probability (assumption).
Penalties λ_1 and λ_2	$50 per trip; $10 per minute beyond the working day	Test weights in equivalent pesos; they should be defined with the logistics manager.

Source: Own elaboration based on company information, CONUEE (2025, 2026) and GasMapa (2026).

Based on these assumptions, Table 6 characterizes the corridors and Figure 3 shows the fuel cost of the isolated round trip to each destination. If each of the 35 orders were delivered on a dedicated trip, 1,692 km would be traveled at a cost of $5,728.63 MXN; however, this value is only a reference, since on a route with several customers consumption is calculated over the arcs actually traveled between consecutive stops. Even so, the comparison shows that an isolated delivery in the metropolitan corridor costs up to 11.8 times more than one in Tequixquiac, which justifies order consolidation.

Table 6. Characterization of the distribution corridors

Corridor	Destinations	One-way distance (km)	One-way time (min)	Demand (kg)	Total service (min)	Mean isolated cost (MXN)
A. Tequixquiac	9	5–8	10–15	219	100	$38.37
B. Zumpango / Zitlaltepec	12	15–24	22–35	282	125	$142.20
C. Metropolitan	14	22–59	32–72	485	210	$262.63
Total	35	5–59	10–72	986	435	$163.68

Note: Isolated cost = fuel for the plant–destination–plant round trip (Equations 1 and 2). Source: Own elaboration.

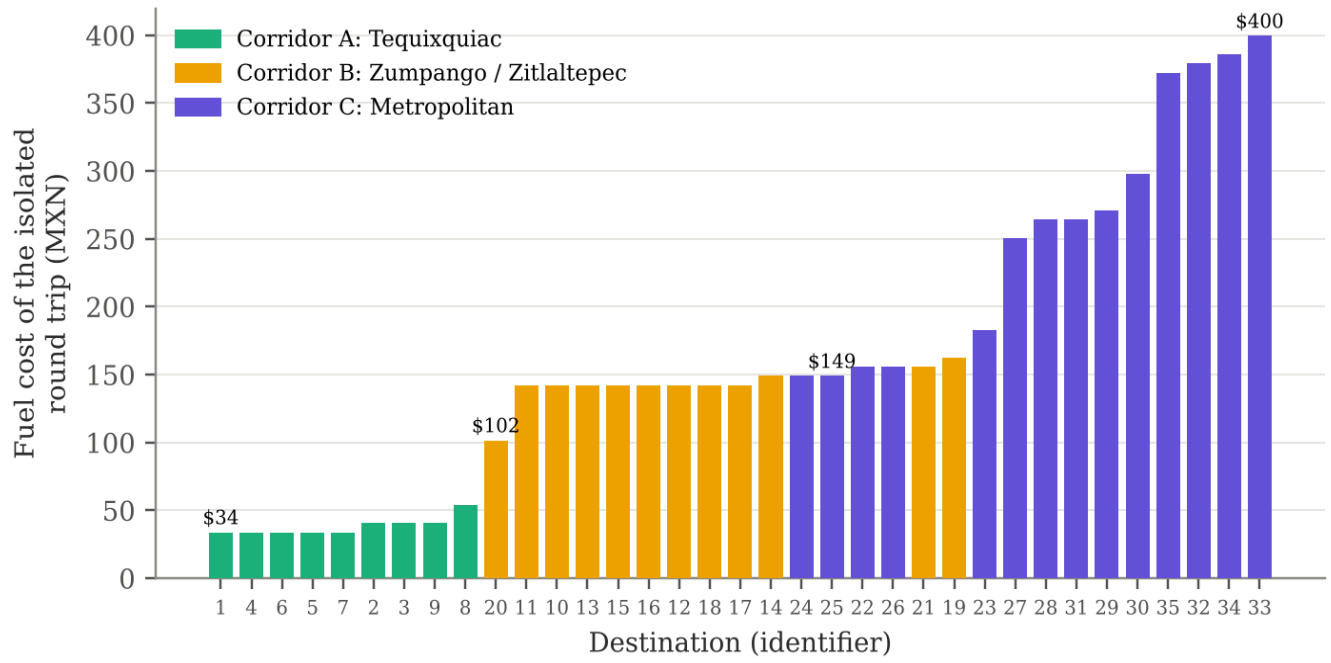


Figure 3. Fuel cost of the isolated round trip by destination and corridor

Source: Own elaboration with one-way distances from the diagnosis, 7 km/L and $23.70 MXN/L.

Phase 2. Define: problem statement and requirements

Based on the understanding gained, the problem was stated as follows: how might we assign and sequence the 35 daily orders between two trucks of 600 and 800 kg, considering that orders become ready at different times of the working day, so as to reduce distance, fuel and returns to the plant without exceeding the 07:00 to 15:00 schedule? This statement recognizes that improvement does not consist only of choosing shorter paths, but also of synchronizing production, order preparation and dispatch.

Next, the identified needs were translated into verifiable requirements, each associated with a quantitative indicator that served as the baseline and as the evaluation criterion (Table 7).

Table 7. Needs, requirements and indicators defined for the model

Code	Identified need	Model requirement	Indicator
R1	High distribution costs and fuel consumption	Minimize the fuel cost of the working day	km; L; MXN per day, per kg and per delivery
R2	Returns to the plant for more product	Represent and minimize additional trips	Number of trips from the plant
R3	Long routes and exceeded working day	Respect the 480-min horizon or penalize its excess	Minutes beyond the working day; return time
R4	Idle capacity on trips	Respect 600 and 800 kg and consolidate the load	Capacity utilization per trip (%)
R5	Orders released during the shift	Prevent departures before the order is ready	Departure time ≥ release time
R6	Accessible tool for an SME	Open, reproducible and low-cost implementation	Computing time; free dependencies
R7	Reliability of the solution	Verify the quality of the routes obtained	Gap to the optimum; variability between runs

Source: Own elaboration based on the problems reported by the company.

Phase 3. Ideate: model selection and formulation

In the ideation phase, three alternatives were compared. First, a static CVRP would solve assignment and sequencing if all orders were ready before departure, but it would not represent returns to the plant (R2) or staggered release (R5). Second, a VRPTW would add the working-day limit (R3), although it would still assume a single trip per vehicle. Finally, a capacitated, heterogeneous, multi-trip VRP with release times (MT-CVRP) satisfies all seven requirements, so it was the selected alternative; the static CVRP was kept as a reference scenario to measure the effect of synchronizing production and dispatch. Table 8 presents the model notation.

Table 8. Sets, parameters and decision variables of the MT-CVRP model

Symbol	Definition
N	Set of customers, indexed by i, j; node 0 represents the plant.
V	Set of vehicles, $v \in \{1, 2\}$, with capacities $Q_1 = 600$ kg and $Q_2 = 800$ kg.
K_v	Set of possible trips of vehicle v during the working day, $k \in \{1, \ldots, \lvert K_v \rvert\}$.
q_i, s_i, r_i	Demand (kg), service time (min) and release time (min) of order i.
d_{ij}, t_{ij}	Distance (km) and driving time (min) of arc (i, j).
c_f, e_v	Fuel price (MXN/L) and fuel economy of vehicle v (km/L).
H, L_c, M	Working-day horizon (480 min), loading time per trip (20 min) and a sufficiently large constant.
λ_1, λ_2	Penalty per activated trip (MXN) and per minute beyond the working day (MXN/min).
$x_{ijvk} \in \{0,1\}$	1 if vehicle v on trip k travels directly from node i to node j.
$y_{ivk} \in \{0,1\}$	1 if customer i is served by vehicle v on trip k.
$w_{vk} \in \{0,1\}$	1 if trip k of vehicle v is used.
S_{vk}, $E_{vk} \geq 0$	Start of loading and return time to the plant of trip k of vehicle v.
$T_{ivk} \geq 0$; $C_v \geq 0$	Arrival time at customer i; end time of the working day of vehicle v.

Source: Own elaboration.

With this notation, the objective function (Equation 3) minimizes fuel cost and penalizes both the activation of additional trips and the time that exceeds the working-day horizon:

$$\text{Min } Z = \Sigma_v \Sigma_k \Sigma_i \Sigma_j \, (c_f / e_v) \, d_{ij} \, x_{ijvk} + \lambda_1 \, \Sigma_v \Sigma_k \, w_{vk} + \lambda_2 \, \Sigma_v \max(0, C_v - H) \tag{3}$$

subject to the following constraints:

$$\Sigma_v \Sigma_k \, y_{ivk} = 1 \quad \forall \, i \in N \tag{4}$$

$$\Sigma_j \, x_{ijvk} = \Sigma_j \, x_{jivk} = y_{ivk} \quad \forall \, i \in N, \, v \in V, \, k \in K_v \tag{5}$$

$$\Sigma_{j \in N} \, x_{0jvk} = \Sigma_{i \in N} \, x_{i0vk} = w_{vk} \quad \forall \, v, \, k \tag{6}$$

$$\Sigma_i \, q_i \, y_{ivk} \leq Q_v \, w_{vk} \quad \forall \, v, \, k \tag{7}$$

$$S_{vk} \geq r_i \, y_{ivk} \quad \forall \, i, \, v, \, k \tag{8}$$

$$T_{jvk} \geq S_{vk} + L_c + t_{0j} - M(1 - x_{0jvk}) \quad \forall \, j, \, v, \, k \tag{9}$$

$$T_{jvk} \geq T_{ivk} + s_i + t_{ij} - M(1 - x_{ijvk}) \quad \forall \, i, j \in N, \, v, \, k \tag{10}$$

$$E_{vk} \geq T_{ivk} + s_i + t_{i0} - M(1 - x_{i0vk}); \quad S_{v,k+1} \geq E_{vk}; \quad C_v \geq E_{vk} \tag{11}$$

$$x_{ijvk}, \, y_{ivk}, \, w_{vk} \in \{0,1\}; \quad S_{vk}, \, E_{vk}, \, T_{ivk}, \, C_v \geq 0 \tag{12}$$

Equation 4 ensures that each order is assigned to exactly one vehicle and trip; Equation 5 links the visit with the entry to and exit from each customer; and Equation 6 ensures that each active trip starts and ends at the plant, which makes it possible to count the returns. In turn, Equation 7 limits the load of each trip to 600 or 800 kg; Equation 8 prevents the loading of a trip from starting before its orders are released; and Equations 9 and 10 propagate arrival times and, at the same time, eliminate subtours. Finally, Equation 11 prevents the overlap of trips of the same vehicle and defines the end time of its working day, while Equation 12 sets the domain of the variables. Note that, when all orders are available at 07:00 ($r_i = 0$), the model reduces to a heterogeneous CVRP with a working-day limit.

Phase 4. Prototype: implementation in Python

The prototype was implemented in Python 3.13 with the NumPy, SciPy and Matplotlib libraries, all open source, in accordance with requirement R6. As shown in Figure 4, the code was organized into four modules: data.py builds the instance; model.py contains the evaluation of the objective function, the savings heuristic, simulated annealing, the rules of the current operation and the exact verification; experiments.py runs the scenarios and analyses; and plots.py generates the figures. The complete code is presented in Appendix B.

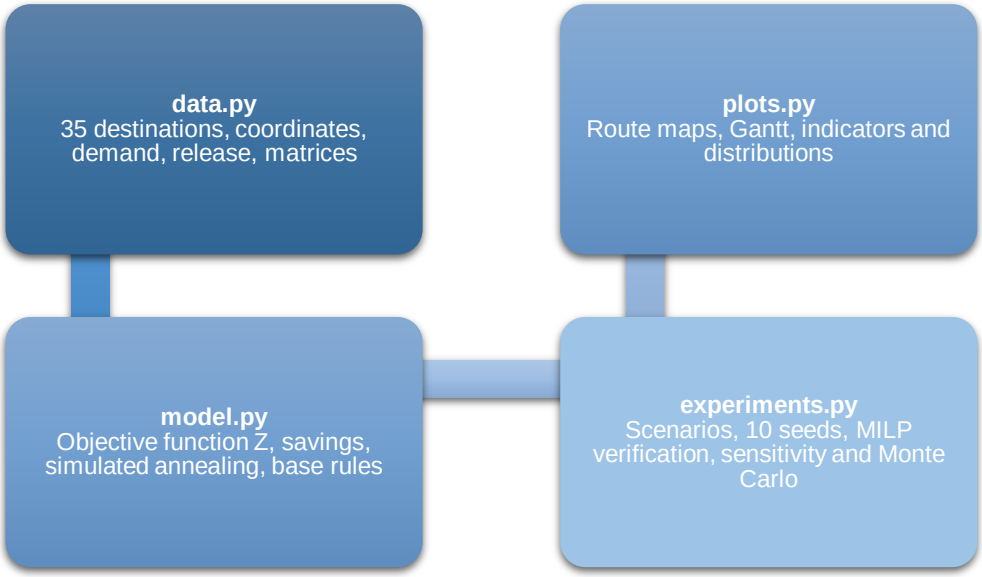


Figure 4. Modular architecture of the VRP prototype developed in Python

Source: Own elaboration.

Distance and time matrices. Since the vehicle visits customers in sequence, the model requires an origin-destination matrix among the 36 nodes and not only the distance from the plant. Consequently, the plant-customer arcs kept the distances and times from the diagnosis, while the arcs between customers were estimated with the geodesic distance δ_{ij} (haversine formula) and a circuity factor k (Equation 13). The median ratio between the reported road distance and the geodesic distance was 0.986; since a road distance cannot be shorter than the geodesic distance, $k = 1.00$ was adopted and its effect was evaluated in the sensitivity analysis. Likewise, driving time was estimated with the linear model of Equation 14, fitted with the 35 distance-time pairs from the diagnosis ($a = 6.21$ min, $b = 1.140$ min/km, $R^2 = 0.994$).

$$d_{ij} = k \cdot \delta_{ij} \tag{13}$$

$$t_{ij} = a + b \cdot d_{ij} \tag{14}$$

Constructive heuristic. The initial solution was obtained with the parallel version of the savings heuristic (Clarke and Wright, 1964), which merges the routes of customers i and j in decreasing order of the saving in Equation 15, provided that the resulting load does not exceed 800 kg. The routes were then assigned as trips to the vehicles, in order of release, to the vehicle with sufficient capacity that would finish its previous trip first.

$$S_{ij} = d_{i0} + d_{0j} - d_{ij} \quad (15)$$

Metaheuristic. The initial solution was improved with simulated annealing (Kirkpatrick et al., 1983), which at each iteration randomly applies one of six moves: relocate a customer (including into a new trip), swap two customers, reverse a segment (2-opt), move a segment of two or three customers (or-opt) or move a complete trip between vehicles. Neighbors that violate capacity are discarded, and feasible ones are accepted with the probability of Equation 16, where the initial temperature equals 2 % of the *Z* value of the savings solution and decreases geometrically to 0.05 over 300,000 iterations. Finally, each trip is polished with an exhaustive 2-opt procedure.

$$P(\text{accept}) = \min\{1, \exp(-\Delta Z / T_n)\}; \quad T_{n+1} = \alpha \cdot T_n \quad (16)$$

Rules of the current operation. To establish a comparable baseline, the dispatching rules described by the company were programmed: each vehicle leaves as soon as released orders exist, serves a single corridor per trip (the one with the largest available load), visits customers in increasing order of distance from the plant along the main connection, and returns for more product. **Exact verification.** Finally, the sequence of each trip obtained was compared with the optimum of the traveling salesman problem over the same nodes, solved as a mixed-integer linear program with iterative subtour elimination using the HiGHS solver included in SciPy (Virtanen et al., 2020).

Phase 5. Test: experimental design

The test phase compared three scenarios with the same orders, vehicles, matrices and assumptions, as detailed in Table 9. For each optimized scenario, 10 independent simulated annealing runs with different seeds were executed, and the best solution was reported together with the mean and standard deviation of Z. In addition, sensitivity analyses were performed with respect to service time, demand, the circuity factor and order release time, as well as a Monte Carlo simulation of 30 working days with random demands and release times. The improvement in each indicator was expressed through Equation 17.

$$\Delta(\%) = [(I_{current} - I_{model}) / I_{current}] \times 100 \quad (17)$$

where $I_{current}$ and I_{model} represent the value of an indicator in the current operation and in the evaluated scenario, respectively.

Table 9. Configuration of the experimental scenarios

Scenario	Order release	Planning method	Purpose
Current operation	Three waves (07:00, 09:30 and 11:30)	Reactive dispatching rules by corridor	Baseline reconstructed from the practice described by the company.
A. Static CVRP	All orders ready at 07:00	Savings + simulated annealing (10 seeds)	Improvement potential with synchronized production and dispatch.
B. MT-CVRP with release	Three waves (07:00, 09:30 and 11:30)	Savings + simulated annealing (10 seeds)	Improvement achievable with the same product availability as the current operation.

Note: All three scenarios used 35 orders (986 kg), two vehicles of 600 and 800 kg, 7 km/L and $23.70 MXN/L.
Source: Own elaboration.

The runs were executed on a Linux server with a general-purpose virtual processor. It should be noted that the study is based on documentary information and computational simulation, so it did not involve the direct participation of people or the collection of personal data.

RESULTS AND DISCUSSION

The results are presented following the order of the Design Thinking test phase. First, the reconstructed baseline of the current operation is described; second, the solutions of scenarios A and B; next, the comparison of indicators, the performance of the algorithm and its exact verification; and, finally, the sensitivity analysis, the Monte Carlo simulation and the fulfillment of the defined requirements.

Reconstructed current operation

With the reactive dispatching rules and order release in three waves, the current operation required 6 trips from the plant, that is, 4 returns to load more product, and traveled 382.0 km with an estimated consumption of 54.6 L and a fuel cost of $1,293.34 MXN. As shown in Figure 5, each trip serves a single corridor along the main connection, so the vehicles repeat the initial stretch from Apaxco several times during the day. Consequently, vehicle 1 finished at 17:13 and vehicle 2 at 17:29, adding up to 282 minutes outside the 07:00 to 15:00 schedule.

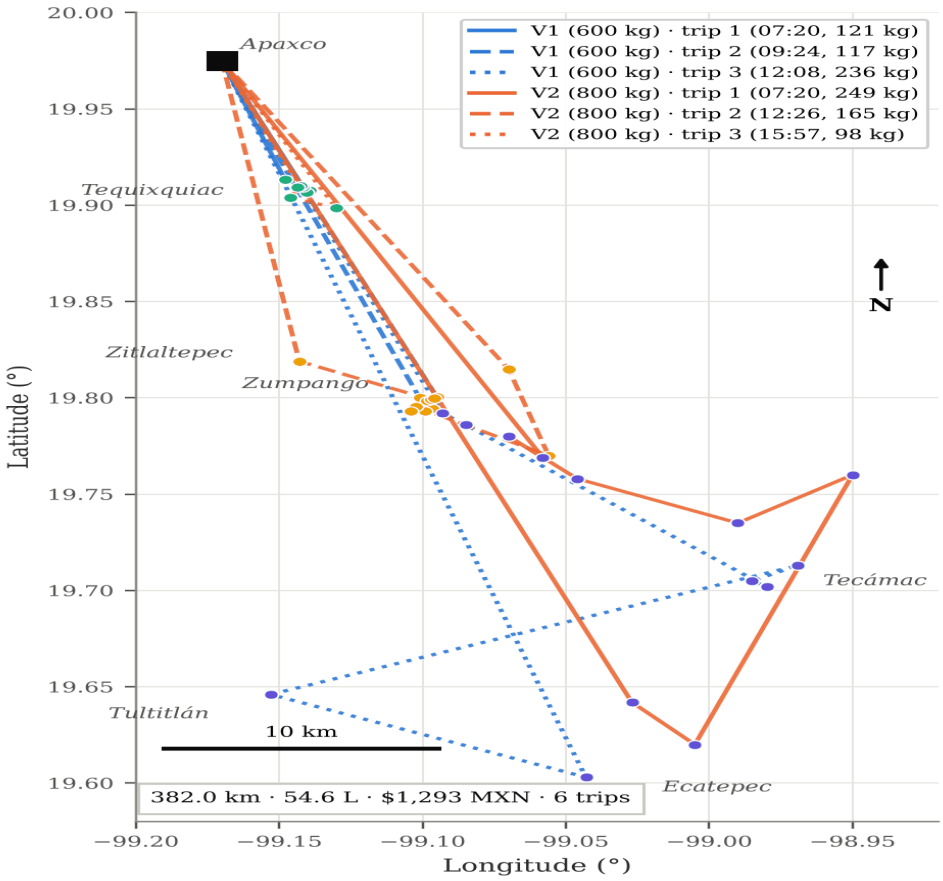


Figure 5. Routes of the reconstructed current operation with reactive dispatching rules

Note: Straight lines represent the visiting sequence, not the road layout. Departure time and trip load are shown in parentheses.

Source: Own elaboration with results from the Python model.

This behavior confirms the company's diagnosis: returns to the plant are not due to a lack of capacity, since the maximum load of a trip was 249 kg against 800 kg available, but to the combination of orders released at different times with a rule that dispatches as soon as goods are available, without evaluating whether it is worth consolidating.

Scenario A: CVRP with synchronized production

When all orders are ready at 07:00, the model solved the distribution with a single trip per vehicle (Figure 6). On the one hand, the 600-kg vehicle served 12 of the 14 destinations of the metropolitan corridor, together with the peripheral destinations 19, 20 and 21 of corridor B and one in Tequixquiac (16 stops and 518 kg); on the other, the 800-kg vehicle served Tequixquiac, Zumpango town center and the two metropolitan destinations located there (19 stops and 468 kg), taking advantage of the fact that Tequixquiac lies on the way between the plant and Zumpango. Thus, the total distance was 152.8 km, the fuel cost $517.30 MXN, and both vehicles returned before 15:00 (14:55 and 13:37).

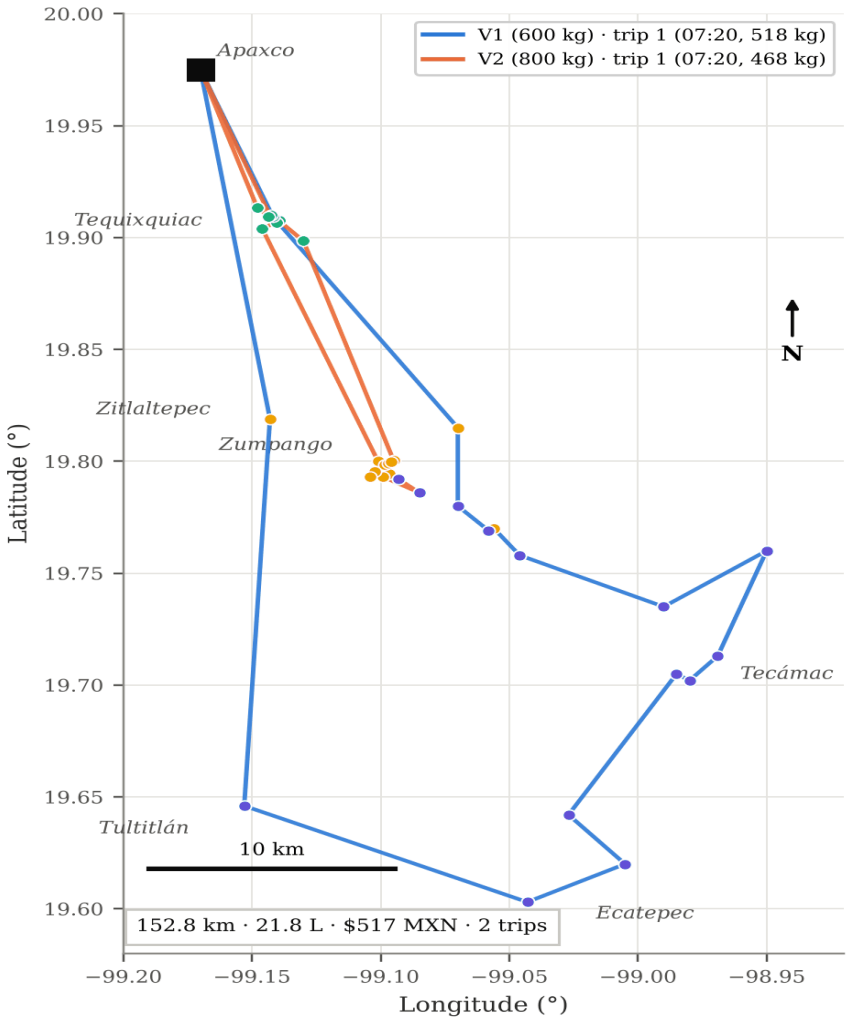


Figure 6. Routes of scenario A (CVRP with all orders ready at 07:00)

Note: Straight lines represent the visiting sequence, not the road layout. Source: Own elaboration with results from the Python model.

Notably, the model assigned the route with the largest load to the lower-capacity vehicle, which is feasible because 518 kg do not exceed 600 kg, and allows the 800-kg vehicle to finish earlier. This result illustrates that the optimal assignment depends on the time and distance of each route, and not only on the nominal capacity of the units.

Scenario B: multiple trips with order release

With the same product availability as the current operation, scenario B reduced trips from 6 to 4 and distance to 285.4 km, with a fuel cost of $966.38 MXN. As shown in Figure 7, the model combined customers from different corridors within the same trip and grouped orders from different waves instead of dispatching each

one separately: the second trip of vehicle 1 departed at 09:53 with 317 kg, and the second trip of vehicle 2 at 11:51 with 283 kg, bringing together orders released at 09:30 and at 11:30. Table 10 details the trips of the three scenarios.

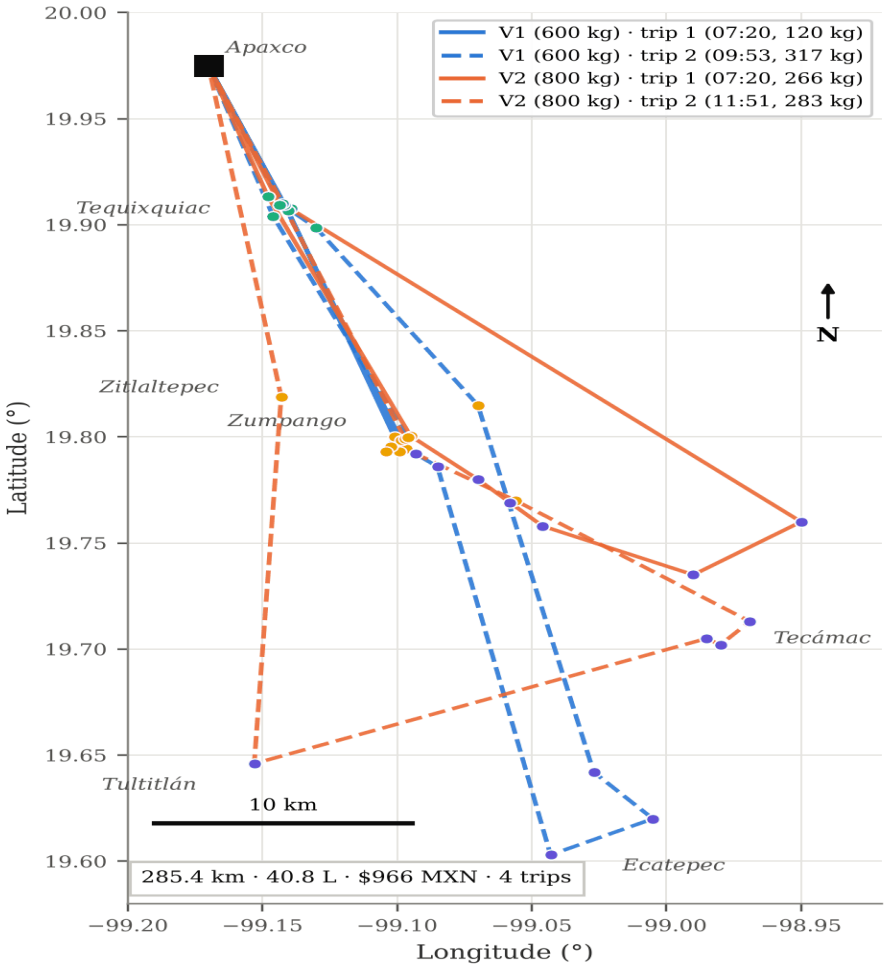


Figure 7. Routes of scenario B (MT-CVRP with staggered order release)

Note: Straight lines represent the visiting sequence, not the road layout. Source: Own elaboration with results from the Python model.

Table 10. Trip details by scenario

Scen.	Veh.	Trip	Depart	Return	Load (kg)	Use (%)	km	Destination sequence
Current	V1	1	07:20	09:04	121	20	13.3	1 → 5 → 7 → 2 → 3
Current	V1	2	09:24	11:48	117	20	46.4	10 → 11 → 12 → 18 → 14
Current	V1	3	12:08	17:13	236	39	132.1	24 → 25 → 28 → 31 → 29 → 35 → 33
Current	V2	1	07:20	12:07	249	31	116.6	22 → 26 → 23 → 27 → 30 → 32 → 34
Current	V2	2	12:27	15:37	165	21	57.5	20 → 13 → 15 → 16 → 17 → 21 → 19
Current	V2	3	15:57	17:29	98	12	16.0	4 → 6 → 9 → 8
A	V1	1	07:20	14:55	518	86	107.8	5 → 19 → 26 → 22 → 21 → 23 → 27 →

								30 → 29 → 28 → 31 → 34 → 32 → 33 → 35 → 20
A	V2	1	07:20	13:37	468	58	45.0	4 → 3 → 9 → 10 → 17 → 18 → 15 → 25 → 24 → 11 → 13 → 16 → 12 → 14 → 8 → 2 → 6 → 7 → 1
B	V1	1	07:20	09:33	120	20	38.4	5 → 12 → 18 → 10 → 3
B	V1	2	09:53	15:00	317	53	88.1	9 → 16 → 17 → 11 → 25 → 33 → 32 → 34 → 19 → 8 → 4
B	V2	1	07:20	11:31	266	33	68.3	7 → 2 → 30 → 27 → 23 → 22 → 26 → 14 → 1
B	V2	2	11:51	16:49	283	35	90.6	6 → 13 → 15 → 24 → 21 → 29 → 28 → 31 → 35 → 20

Note: The numbers in the sequence correspond to the identifiers in Appendix A; each trip starts and ends at the plant.
Source: Own elaboration.

However, scenario B did not completely eliminate time beyond the working day: vehicle 2 returned at 16:49, with 109 minutes of excess, because the orders of the 11:30 wave include destinations in Tecámac and Tultitlán that cannot be served within the schedule once released. Far from being a limitation of the model, this finding reveals that the dominant constraint of the system is the production release time, an aspect analyzed in more detail in the sensitivity section.

Comparison of indicators

Table 11 and Figure 8 summarize the indicators of the three scenarios. Compared with the current operation, scenario B reduced distance and fuel cost by 25.3 %, eliminated 2 of the 4 returns to the plant and decreased the minutes beyond the working day by 61.2 %. In unit terms, the fuel cost per delivery fell from $36.95 to $27.61 MXN and the cost per kilogram from $1.31 to $0.98 MXN. Scenario A, in turn, achieved a 60.0 % reduction in distance and cost, with no returns and no time beyond the working day.

Table 11. Performance indicators of the current operation and of the optimized scenarios

Indicator	**Current operation**	**Scenario A**	**Scenario B**	**Δ A (%)**	**Δ B (%)**
Distance traveled (km)	382.0	152.8	285.4	60.0	25.3
Fuel (L)	54.6	21.8	40.8	60.0	25.3
Fuel cost (MXN)	$1,293.34	$517.30	$966.38	60.0	25.3
Trips from the plant	6	2	4	66.7	33.3
Returns for more product	4	0	2	100.0	50.0
Minutes beyond the working day	282	0	109	100.0	61.2
End time V1 / V2	17:13 / 17:29	14:55 / 13:37	15:00 / 16:49	–	–
Kilometers per delivery	10.91	4.37	8.16	60.0	25.3
Fuel cost per kg (MXN)	1.31	0.52	0.98	60.0	25.3
Fuel cost per delivery (MXN)	36.95	14.78	27.61	60.0	25.3
Mean utilization per trip (%)	23.8	72.4	35.4	–	–
Objective function Z (equivalent MXN)	4,415.34	617.30	2,261.38	86.0	48.8

Note: Δ calculated with Equation 17; positive values indicate reduction. Z includes fuel, λ_1 = $50 per trip and λ_2 = $10 per minute beyond the working day.

Source: Own elaboration.

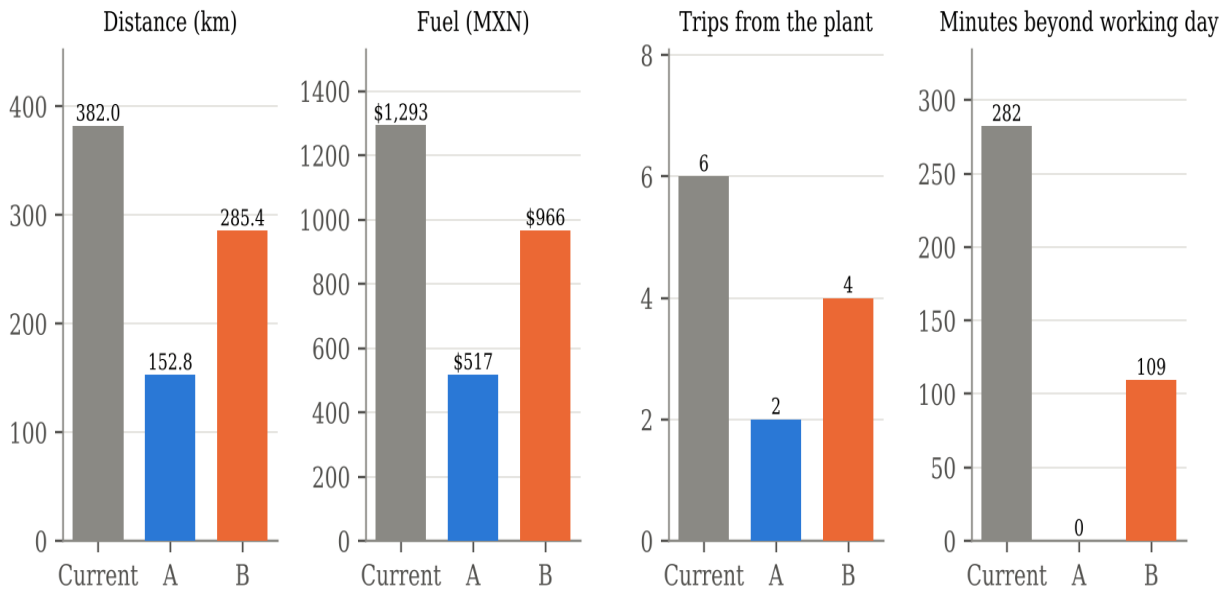


Figure 8. Comparison of indicators between the current operation and scenarios A and B

Source: Own elaboration with results from the Python model.

From a time perspective, Figure 9 shows the schedule of each vehicle. In the current operation, the gray loading blocks recur throughout the day and the third trip of vehicle 2 starts after 15:00; in contrast, scenario A concentrates distribution in one continuous block per vehicle and scenario B reduces departures to two per unit. Regarding fleet utilization, Figure 10 indicates that the mean utilization per trip rose from 23.8 % in the current operation to 35.4 % in scenario B and to 72.4 % in scenario A, which confirms that order consolidation is the main savings mechanism.

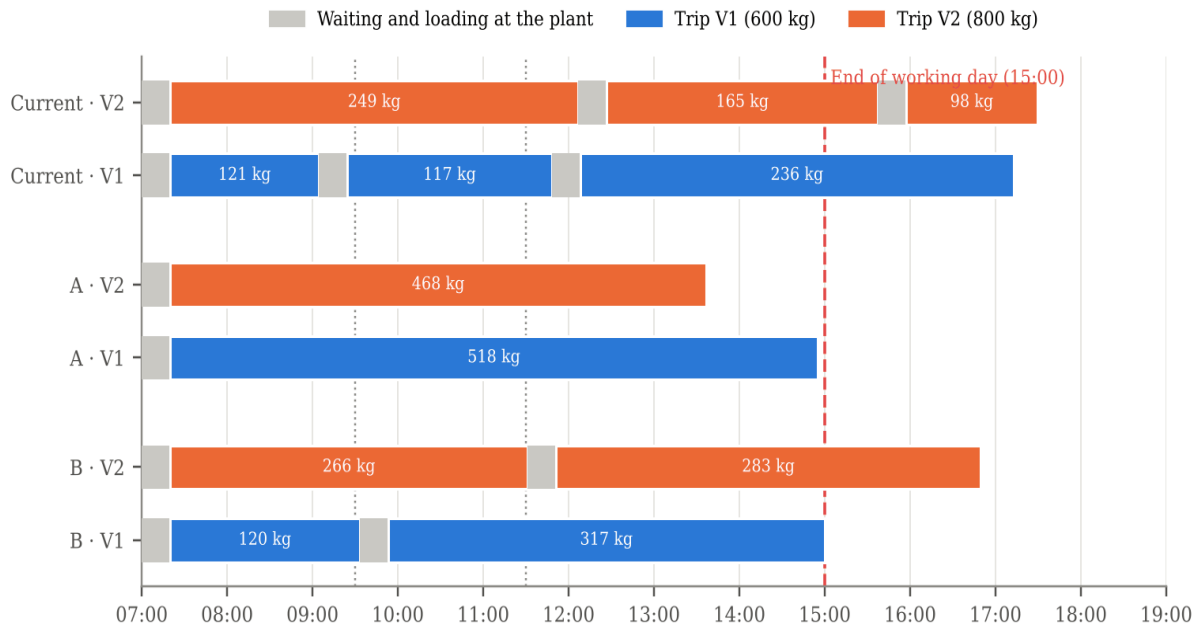


Figure 9. Trip schedule by vehicle and scenario

Note: The vertical dotted lines indicate the release of the 09:30 and 11:30 waves.
Source: Own elaboration.

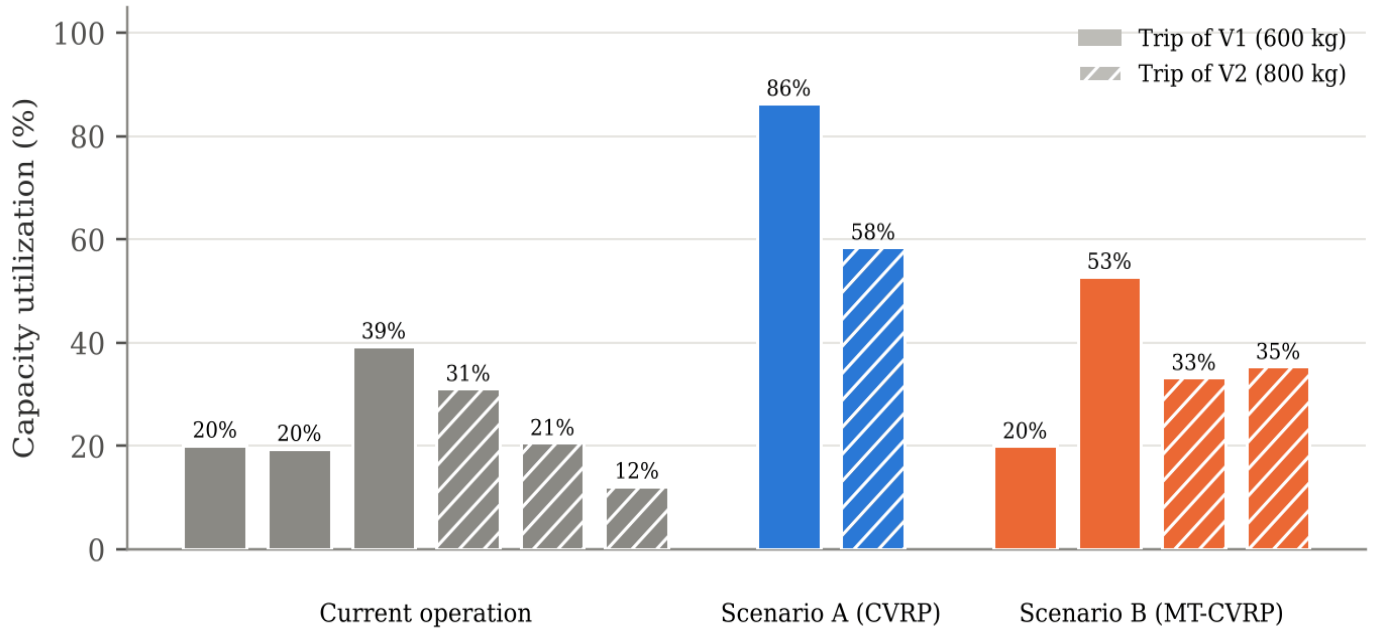


Figure 10. Capacity utilization per trip in each scenario

Source: Own elaboration.

Algorithm performance and exact verification

Table 12 summarizes the computational performance. The savings heuristic minimizes distance, but since it does not consider the working-day horizon it produces solutions with considerable time excess; therefore, simulated annealing improved the value of *Z* by 76.8 % in scenario A and by 57.8 % in scenario B. Likewise, the algorithm showed low variability across seeds, with coefficients of variation of 2.86 % and 1.29 %, and a mean time per run of 10.8 and 13.1 s, compatible with daily planning on a conventional computer.

Table 12. Algorithm performance in 10 independent runs and exact verification of the sequences

Scenario	Z savings	Best Z	Mean Z ± SD	CV (%)	Improvement vs. savings (%)	Runs reaching the best	Mean time (s)	Trips with optimal sequence
A. Static CVRP	2,658.27	617.30	623.25 ± 17.83	2.86	76.8	9/10	10.8	2/2
B. MT-CVRP with release	5,358.27	2,261.38	2,294.56 ± 29.53	1.29	57.8	1/10	13.1	4/4

Note: SD = standard deviation; CV = coefficient of variation. The optimal sequence was verified with a traveling salesman MILP model solved with HiGHS.
Source: Own elaboration.

Figure 11 presents the convergence of the best run. In both scenarios, most of the improvement is obtained during the first 100,000 iterations and the solution stabilizes before 200,000, which indicates that the number of iterations set is sufficient for the size of the instance.

In addition, the exact verification confirmed that the sequence of the six trips of scenarios A and B matches the optimum of the traveling salesman problem over the same customers; in other words, once customers are assigned to each trip, there is no visiting order with a shorter distance, which meets reliability requirement R7.

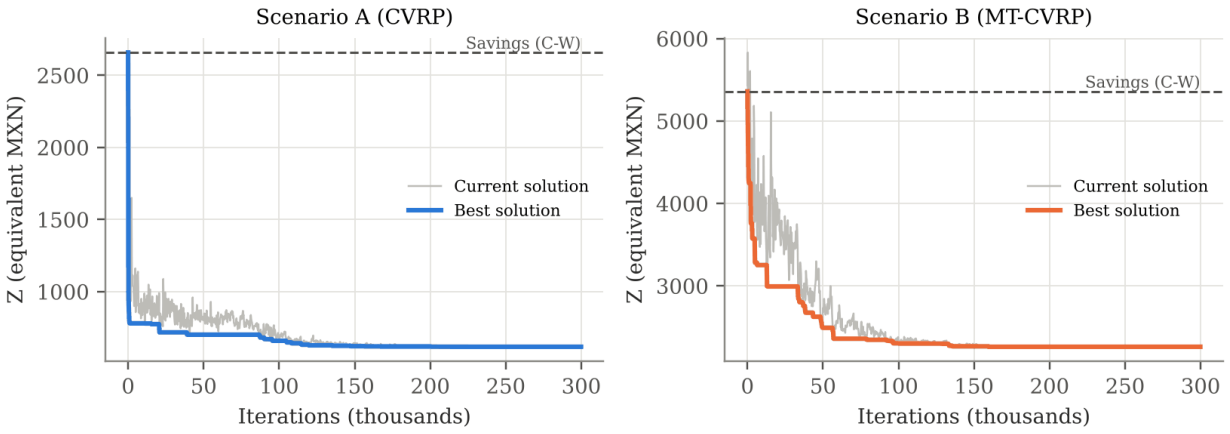


Figure 11. Convergence of simulated annealing in scenarios A and B

Note: The dashed line indicates the Z value of the initial savings solution (Clarke and Wright, 1964). Source: Own elaboration.

Sensitivity analysis

The sensitivity analysis is summarized in Table 13 and Figure 12. The most influential factor was the production release time: if the last wave is brought forward from 11:30 to 08:30, the fuel cost of the optimized scenario drops from $966.38 to $611.46 MXN and the time beyond the working day from 110 to 15 minutes; in contrast, the current operation barely improves, because its rules keep dispatching by corridor. In other words, the value of synchronizing production is fully captured only when it is accompanied by optimized route planning.

Table 13. Sensitivity of fuel cost, trips and time beyond the working day

Parameter evaluated	Current cost (MXN)	Optimized cost (MXN)	Reduction (%)	Trips current / opt.	Min. beyond working day current / opt.
Last wave at 11:30 (base)	1,293.34	966.38	25.3	6 / 4	282 / 110
Last wave at 10:30	1,284.30	819.78	36.2	6 / 4	278 / 71
Last wave at 09:30	1,284.30	768.66	40.1	6 / 4	278 / 43
Last wave at 08:30	1,145.79	611.46	46.6	5 / 4	203 / 15
Last wave at 07:00 (all ready)	800.55	517.30	35.4	3 / 2	38 / 0
Service ×0.8 (8/12 min)	1,598.40	975.73	39.0	–	324 / 46
Service ×1.0 (10/15 min)	1,293.34	966.38	25.3	–	282 / 110
Service ×1.2 (12/18 min)	1,293.34	845.82	34.6	–	369 / 181
Service ×1.4 (14/21 min)	1,293.34	792.63	38.7	–	456 / 239
Circuity factor k = 1.00	1,293.34	966.38	25.3	–	282 / 110
Circuity factor k = 1.15	1,350.90	959.38	29.0	–	302 / 143
Circuity factor k = 1.30	1,408.52	1,009.72	28.3	–	321 / 159
Demand ×1.0 (986 kg)	1,293.34	966.38	25.3	6 / 4	282 / 110
Demand ×1.1 (1,085 kg)	1,293.34	970.11	25.0	6 / 4	282 / 113
Demand ×1.2 (1,183 kg)	1,293.34	853.94	34.0	6 / 4	282 / 122
Demand ×1.3 (1,281 kg)	1,293.34	970.11	25.0	6 / 4	282 / 113
Demand ×1.4 (1,379 kg)	1,293.34	966.52	25.3	6 / 4	282 / 110

Note: Each optimized configuration corresponds to the best of two runs of 150,000 iterations; shaded rows are the base configuration. Cost = fuel (Equations 1 and 2). Source: Own elaboration.

Regarding service time, increasing it raises the time beyond the working day in both schemes, although the model systematically keeps it below that of the current operation. Notably, with shorter service times (8 and 12 min) the current operation worsens, because vehicles return earlier, find newly released orders and make more trips with little load (472.1 km); this non-monotonic behavior is typical of reactive rules and disappears with optimized planning. As for the circuity factor, increasing *k* from 1.00 to 1.30 raises the costs of both schemes, but the relative reduction remains between 25.3 and 29.0 %. Finally, demand growth up to 1,379 kg did not change the number of trips, which confirms that the aggregate capacity of 1,400 kg is not the active constraint of the system.

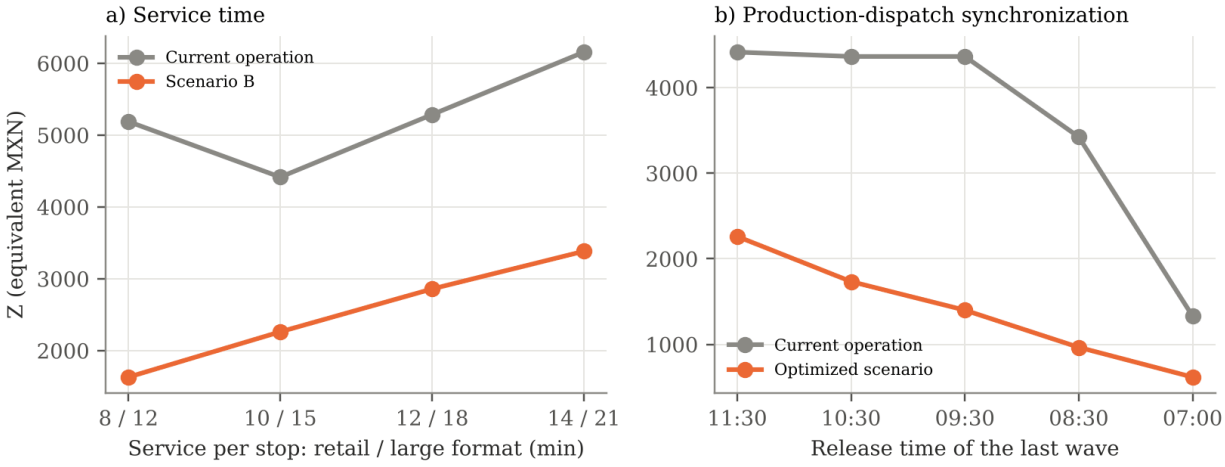


Figure 12. Sensitivity of the objective function to service time and to the production release time

Source: Own elaboration with results from the Python model.

Robustness to daily variability

To verify that the results do not depend on a single working day, 30 days were simulated with demands between 967 and 1005 kg and random assignments of orders to the production waves. As shown in Figure 13, optimized planning reduced distance on every simulated day, with a mean of 35.1 % (standard deviation of 5.1 %; range from 25.0 to 45.6 %), and the objective function by 56.1 % on average. Likewise, it avoided between 1 and 4 trips per day (mode of 2), reduced the mean time beyond the working day from 343 to 112 minutes and generated mean fuel savings of $509.28 MXN per day. If these savings are extrapolated to 24 working days per month, they amount to approximately $12,223 MXN per month in fuel alone, without considering maintenance, wear or staff overtime.

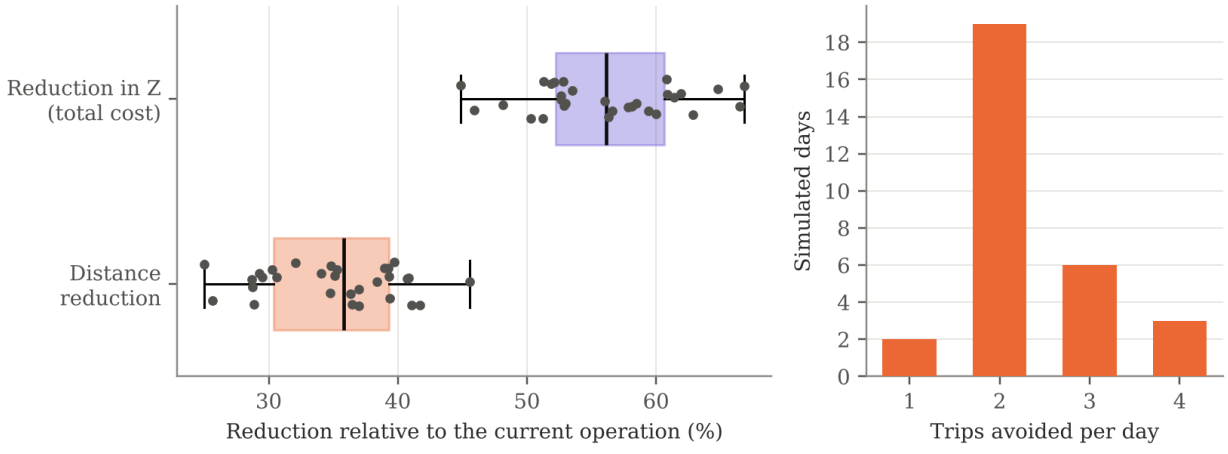


Figure 13. Distribution of the reductions and of the trips avoided over 30 simulated working days

Source: Own elaboration with results from the Monte Carlo simulation.

Discussion and fulfillment of requirements

The results support the proposed hypothesis: planning with a multi-trip VRP model with order release reduces distance, fuel and returns to the plant compared with the reactive rules, and the magnitude of the reduction depends on the synchronization between production and dispatch. In relation to the literature, the 25.3 % reduction in scenario B and the 35.1 % average reduction in the Monte Carlo simulation are of the same order as the 41 % economic improvement reported by González-Romero et al. (2026) for a multi-trip model, while the decisive role of loading time and trip duration is consistent with Pan et al. (2021). Similarly, as in the dairy case of Sanabria-Rey et al. (2021), a simple, low-cost tool proved sufficient to achieve substantial improvements in a company without specialized technological infrastructure.

Moreover, the study provides a finding that the reviewed literature does not clearly highlight for SMEs that produce and dispatch on the same day: route optimization by itself has a ceiling determined by the order release time. While scenario B, with current production, achieves a 25.3 % reduction in distance, scenario A, with synchronized production, reaches 60.0 %. Consequently, the managerial recommendation is not limited to adopting the model, but also includes scheduling the production of the metropolitan corridor orders in the first hours of the working day, since they have the highest isolated cost and are the ones that generate time beyond the working day.

Finally, closing the Design Thinking cycle, Table 14 contrasts the requirements defined in Table 7 with the evidence obtained. Five requirements were fully met with the prototype, while the working-day requirement (R3) was partially met in scenario B and the consolidation requirement (R4) remained conditional on the synchronization of production; both define the next iteration of the process.

Table 14. Fulfillment of the requirements defined in the define phase

Code	Requirement	Evidence obtained	Status
R1	Minimize fuel cost	−25.3 % (B) and −60.0 % (A); −35.1 % on average in the Monte Carlo simulation.	Met
R2	Minimize additional trips	Returns to the plant from 4 to 2 (B) and to 0 (A).	Met
R3	Respect the 480-min working day	Excess from 282 to 109 min (B); 0 min (A).	Partial
R4	Consolidate the load	Mean utilization from 23.8 % to 35.4 % (B) and 72.4 % (A).	Conditional
R5	Respect order release	Constraint (8) incorporated; no departure before release.	Met
R6	Accessible tool	Python with free libraries; 13.1 s per run.	Met
R7	Reliability of the solution	Optimal sequences verified (6/6); CV of 2.86 % (A) and 1.29 % (B) across seeds.	Met

Source: Own elaboration.

Nevertheless, the results should be interpreted with caution. Distances between customers were estimated with approximate coordinates and a circuity factor, the fuel economy of 7 km/L is an assumption rather than a measurement, and the portfolio of 35 destinations corresponds to public prospects rather than the actual customer register. Likewise, the current operation was reconstructed from rules described by the company and not from field records. Therefore, the figures represent planning estimates whose validation requires a log of orders, kilograms, schedules, odometer readings and fuel, for which the format in Appendix C is proposed.

CONCLUSION

The study developed and evaluated vehicle routing models implemented in Python for the last-mile distribution of a cheese-producing SME that manufactures and dispatches during the same working day. Under the Design Thinking methodology, understanding the operation made it possible to define seven verifiable requirements, ideate a capacitated, heterogeneous, multi-trip model with order release times, prototype it with

open tools and test it against a reconstruction of the current dispatching rules. Accordingly, the stated objective is considered fulfilled for this stage of the research.

With the same product availability as the current operation, the model reduced the daily distance from 382.0 to 285.4 km and the fuel cost from $1,293.34 to $966.38 MXN (25.3 %), decreased trips from 6 to 4 and reduced the time beyond the working day by 61.2 %. In addition, the simulation of 30 working days confirmed a mean reduction of 35.1 % in distance, with mean fuel savings of $509.28 MXN per day, and the exact verification showed that the sequence of each trip is optimal. Therefore, the hypothesis that planning with the proposed model outperforms the reactive rules is accepted.

However, the finding of greatest managerial relevance is that the benefit of optimization is bounded by the synchronization between production and dispatch: with all orders ready at 07:00, the reduction reaches 60.0 % with a single trip per vehicle and no time beyond the working day. Thus, the company is advised to schedule the production of the metropolitan corridor orders first, to use the model to plan the daily assignment and visiting sequence, and to evaluate the visit frequency by zone as the next decision level.

The limitations include that distances between customers were estimated with approximate coordinates, that fuel economy and service times are assumptions, that the destinations correspond to a potential portfolio and that the current operation was reconstructed from rules described by the company. Therefore, the following lines of future work are proposed: keeping, for at least two weeks, a log of orders, kilograms, schedules, odometer readings and fuel; replacing the estimated matrices with road matrices obtained from a navigation service; incorporating customer receiving windows, storage conditions for dairy products and maintenance and staff costs; and extending the model to a visit calendar by zone every other day. In this way, the Design Thinking cycle can be iterated with field data until the tool is implemented in daily operations.

ACKNOWLEDGEMENTS

The authors thank the participating company for the information provided on its distribution operation, as well as their home institution for the support given to the development of the project.

Ethical Approval

This study did not involve human or animal participants and was based on operational information provided by the company and on computational simulation; therefore, it did not require approval from an ethics committee. At the company's request, its name and those of its potential customers are kept confidential.

Conflict Of Interest

The authors declare that there is no conflict of interest regarding the research, authorship or publication of this article.

Data Availability

The model parameters and instance data (destinations, one-way distances and times, demands, release times and service times) are presented in Tables 4 to 6 and in Appendix A, and the complete Python code that reproduces the results is included in Appendix B. The reference coordinates and the detailed results of each run are available from the corresponding author upon reasonable request, preserving the confidentiality of the participating company.

REFERENCES

1. Álvarez-Miranda, E., Epstein, R., Pereira, J., Sinnl, M., & Urrutia, R. (2025). A multi-criteria districting approach with a lexicographic compactness metric: An application to the Chilean postal service. Computers & Operations Research, 173, 106845. https://doi.org/10.1016/j.cor.2024.106845
2. Brown, T. (2008). Design thinking. Harvard Business Review, 86(6), 84–92.

3. Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. Operations Research, 12(4), 568–581. https://doi.org/10.1287/opre.12.4.568
4. Comisión Nacional para el Uso Eficiente de la Energía. (2025, March 21). Rendimiento de combustible en vehículos ligeros de venta en México [Fuel economy of light-duty vehicles sold in Mexico]. Government of Mexico. https://www.gob.mx/conuee/documentos/rendimiento-de-combustible-en-vehiculos-ligeros-de-venta-en-mexico
5. Comisión Nacional para el Uso Eficiente de la Energía. (2026). Catálogo de rendimiento de combustible en vehículos ligeros de venta en México [Fuel economy catalog of light-duty vehicles sold in Mexico] [Data set]. Datos.gob.mx. https://www.datos.gob.mx/dataset/catalogo_rendimiento_combustible_vehiculos_ligeros_venta_mexico
6. Dahlan, A. F., Cheng, C.-Y., Sae-chai, P., Jientrakul, R., & Yuangyai, C. (2025). Crowdsourced adaptive vehicle routing framework for last-mile delivery in dynamic traffic environments. Annals of Operations Research. https://doi.org/10.1007/s10479-025-06806-2
7. Dantzig, G. B., & Ramser, J. H. (1959). The truck dispatching problem. Management Science, 6(1), 80–91. https://doi.org/10.1287/mnsc.6.1.80
8. Dribel, O., Ait El Cadi, A., Riane, F., & Benmansour, R. (2025). Optimizing last-mile delivery with crowdshipping and relay points: A tabu search approach. IFAC-PapersOnLine, 59(10), 3047–3052. https://doi.org/10.1016/j.ifacol.2025.09.512
9. Errami, N., Queiroga, E., Sadykov, R., & Uchoa, E. (2024). VRPSolverEasy: A Python library for the exact solution of a rich vehicle routing problem. INFORMS Journal on Computing, 36(4), 956–965. https://doi.org/10.1287/ijoc.2023.0103
10. Fonseca-Galindo, J. C., Surita, G. de C., Maia Neto, J., de Castro, C. L., & Lemos, A. P. (2022). A multi-agent system for solving the dynamic capacitated vehicle routing problem with stochastic customers using trajectory data mining. Expert Systems with Applications, 195, 116602. https://doi.org/10.1016/j.eswa.2022.116602
11. GasMapa. (2026, October 20). Precios de combustibles en México [Fuel prices in Mexico]. https://www.gasmapa.com/
12. González-Romero, I., Bastero-Sellán, J., Comesaña Benavides, J. A., & Prado Prado, J. C. (2026). Economic, environmental, and social sustainability in last-mile delivery: Solving the multi-trip vehicle routing problem. Central European Journal of Operations Research, 34(4), 1237–1260. https://doi.org/10.1007/s10100-026-01017-x
13. Google for Developers. (n.d.-a). Capacity constraints. OR-Tools. https://developers.google.com/optimization/routing/cvrp
14. Google for Developers. (n.d.-b). Vehicle routing problem. OR-Tools. https://developers.google.com/optimization/routing/vrp
15. Google for Developers. (n.d.-c). Vehicle routing problem with time windows. OR-Tools. https://developers.google.com/optimization/routing/vrptw
16. Guiamexican. (n.d.). Leche, productos lácteos, embutidos en Zumpango [Milk, dairy products and cold cuts in Zumpango]. https://guiamexican.com/leche-productos-lacteos-embutidos/zumpango
17. Hasso Plattner Institute of Design at Stanford. (2010). An introduction to design thinking: Process guide. Stanford University.
18. He, P., Li, J., Jiang, Y., & Wang, X. (2018). Collaborative transportation planning distinguishing old and new shippers for small-medium enterprise. Journal of Industrial and Production Engineering, 35(3), 170–180. https://doi.org/10.1080/21681015.2018.1443165
19. Hsiao, Y.-H., Chen, M.-C., Lu, K.-Y., & Chin, C.-L. (2018). Last-mile distribution planning for fruit-and-vegetable cold chains. The International Journal of Logistics Management, 29(3), 862–886. https://doi.org/10.1108/IJLM-01-2017-0002
20. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671–680. https://doi.org/10.1126/science.220.4598.671
21. Klein, P. S., & Schiffer, M. (2024). RoutingBlocks: An open-source Python package for vehicle routing problems with intermediate stops. INFORMS Journal on Computing, 36(4), 966–973. https://doi.org/10.1287/ijoc.2023.0104

22. Li, Y., & Yang, J. (2024). The last-mile delivery vehicle routing problem with handling cost in the front warehouse mode. Computers & Industrial Engineering, 190, 110076. https://doi.org/10.1016/j.cie.2024.110076
23. Liao, S., Dai, Y., & Ma, Z. (2025). Vehicle routing problem with roaming locations for simultaneous pickup and delivery services. OPSEARCH, 63(2), 1172–1192. https://doi.org/10.1007/s12597-025-00934-w
24. Maghfiroh, M. F. N., & Hanaoka, S. (2018). Dynamic truck and trailer routing problem for last mile distribution in disaster response. Journal of Humanitarian Logistics and Supply Chain Management, 8(2), 252–278. https://doi.org/10.1108/JHLSCM-10-2017-0050
25. Malik, S., Khonji, M., Elbassioni, K., & Dias, J. M. M. (2025). CODEL: Collaborative multi-robot parcel delivery for last-mile logistics. In Advances in Service and Industrial Robotics (Mechanisms and Machine Science, pp. 706–714). Springer. https://doi.org/10.1007/978-3-032-02106-9_78
26. MXFirmas. (n.d.). Tiendas de lácteos en Zumpango de Ocampo, Estado de México [Dairy stores in Zumpango de Ocampo, State of Mexico]. https://mxfirmas.com/EMX-Zumpango-de-Ocampo/Tiendas-de-lacteos
27. Nieto de Almeida, A., Crespo-Pereira, D., & García del Valle, A. (2012). ELOCONS: un algoritmo de construcción de rutas eficiente para la pequeña y mediana empresa de distribución [ELOCONS: An efficient route construction algorithm for small and medium-sized distribution companies]. DYNA Ingeniería e Industria, 87(3), 222–228. https://doi.org/10.6036/4360
28. Ortuño, M. T., Ferrer, J. M., Flores, I., & Tirado, G. (2021). Optimization of humanitarian logistics. In International Encyclopedia of Transportation (pp. 195–200). Elsevier. https://doi.org/10.1016/B978-0-08-102671-7.10243-X
29. Pan, B., Zhang, Z., & Lim, A. (2021). Multi-trip time-dependent vehicle routing problem with time windows. European Journal of Operational Research, 291(1), 218–231. https://doi.org/10.1016/j.ejor.2020.09.022
30. Rashidi, M. H., Nourinejad, M., & Roorda, M. J. (2025). Generating practical last-mile delivery routes using a data-informed insertion heuristic. Transportation Research Part C: Emerging Technologies, 179, 105278. https://doi.org/10.1016/j.trc.2025.105278
31. Sanabria-Rey, J. G., Solano-Charris, E. L., Vega-Mejía, C. A., & Quintero-Araujo, C. L. (2021). Solving last-mile deliveries for dairy products using a biased randomization-based spreadsheet. A case study. American Journal of Mathematical and Management Sciences, 41(1), 51–69. https://doi.org/10.1080/01966324.2021.1910886
32. SteadieSeifi, M., Dellaert, N., Nuijten, W., & Van Woensel, T. (2017). A metaheuristic for the multimodal network flow problem with product quality preservation and empty repositioning. Transportation Research Part B: Methodological, 106, 321–344. https://doi.org/10.1016/j.trb.2017.07.007
33. Thipparthy, K. R., Khalaf, M., Yogi, K. S., Alghayadh, F. Y., Yusupov, A., Maguluri, L. P., & Ofori-Amanfo, P. (2024). Optimizing delivery routes for sustainable food delivery for multiple food items per order. Discover Sustainability, 5(1). https://doi.org/10.1007/s43621-024-00326-y
34. Utama, D. M., Dewi, S. K., Wahid, A., & Santoso, I. (2020). The vehicle routing problem for perishable goods: A systematic review. Cogent Engineering, 7(1), 1816148. https://doi.org/10.1080/23311916.2020.1816148
35. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2
36. Yang, K., Zhang, Z., Zhao, J., & Liang, Z. (2025). A neighborhood-based matheuristic for the vehicle routing problem with delivery options. Transportation Research Part E: Logistics and Transportation Review, 203, 104366. https://doi.org/10.1016/j.tre.2025.104366
37. Yu, V. F., Lin, S.-W., Zhou, L., & Baldacci, R. (2024). A fast simulated annealing heuristic for the multi-depot two-echelon vehicle routing problem with delivery options. Transportation Letters, 16(8), 921–932. https://doi.org/10.1080/19427867.2023.2257923
38. Yu, X., Li, Y., Liu, J., & Zhou, H. (2024). Solving low-carbon last mile delivery problem using discrete marine predators algorithm. Applied Soft Computing, 165, 112112. https://doi.org/10.1016/j.asoc.2024.112112

39. Zhou, L., Baldacci, R., Vigo, D., & Wang, X. (2018). A multi-depot two-echelon vehicle routing problem with delivery options arising in the last mile distribution. European Journal of Operational Research, 265(2), 765–778. https://doi.org/10.1016/j.ejor.2017.08.011

APPENDICES

Appendix A. Parameters of the 35 potential destinations used in the model

Table A1. Destinations, distances, demand, release and service

ID	Corr.	Business type	Locality	One-way (km)	One-way (min)	Isolated cost	Demand (kg)	Release	Service (min)
1	A	Dairy shop / mini-market	Tequixquiac	5	10	$33.86	21	07:00	10
2	A	Grocery / mini-market	Tequixquiac	6	12	$40.63	23	07:00	10
3	A	Grocery	Tequixquiac	6	12	$40.63	22	07:00	10
4	A	Self-service store	Tequixquiac	5	10	$33.86	30	09:30	15
5	A	Supermarket	Tequixquiac	5	10	$33.86	30	07:00	15
6	A	Mini-market / grocery	Tequixquiac	5	10	$33.86	24	09:30	10
7	A	Cheese shop / dairy	Tequixquiac	5	10	$33.86	25	07:00	10
8	A	Mini-market	Tequixquiac	8	15	$54.17	21	09:30	10
9	A	Producer / cheese shop	Tequixquiac	6	12	$40.63	23	09:30	10
10	B	Dairy shop	Zumpango	21	32	$142.20	21	07:00	10
11	B	Dairy shop	Zumpango	21	32	$142.20	25	07:00	10
12	B	Dairy shop	Zumpango	21	32	$142.20	25	07:00	10
13	B	Dairy shop	Zumpango	21	32	$142.20	23	11:30	10
14	B	Dairy shop / butcher	Zumpango	22	33	$148.97	24	07:00	10
15	B	Dairy shop	Zumpango	21	32	$142.20	23	11:30	10
16	B	Dairy shop	Zumpango	21	32	$142.20	24	09:30	10
17	B	Dairy shop / grocery	Zumpango	21	32	$142.20	22	09:30	10
18	B	Dairy shop / dairy products	Zumpango	21	32	$142.20	22	07:00	10
19	B	Dairy store	San Miguel Bocanegra	24	35	$162.51	21	09:30	10
20	B	Dairy store	San Juan Zitlaltepec	15	22	$101.57	21	09:30	10
21	B	Self-service store	Zumpango (Buenavista)	23	33	$155.74	31	11:30	15
22	C	Warehouse club	Zumpango (Buenavista)	23	33	$155.74	31	07:00	15
23	C	Self-service store	Paseos de San Juan	27	38	$182.83	32	07:00	15
24	C	Self-service store	Zumpango (Bo. Santiago)	22	32	$148.97	28	11:30	15
25	C	Wholesale / grocery	Zumpango (Bo. San Lorenzo)	22	32	$148.97	42	09:30	15
26	C	Wholesale / grocery	San Sebastián	23	34	$155.74	41	07:00	15
27	C	Self-service store	Tecámac (Sierra	37	48	$250.54	29	07:00	15

			Hermosa)						
28	C	Self-service store	Tecámac (Hueyotenco)	39	50	$264.09	32	09:30	15
29	C	Supermarket	Tecámac (Center)	40	50	$270.86	31	11:30	15
30	C	Wholesale market	Tecámac (Ajoloapan)	44	55	$297.94	40	07:00	15
31	C	Warehouse club	Tecámac	39	50	$264.09	29	09:30	15
32	C	Self-service store	Ecatepec	56	70	$379.20	33	07:00	15
33	C	Hypermarket	Ecatepec	59	72	$399.51	33	09:30	15
34	C	Dairy shop / wholesaler	Ecatepec	57	70	$385.97	43	07:00	15
35	C	Wholesale market	Tultitlán	55	68	$372.43	41	11:30	15

Note: Establishments are identified by number to preserve confidentiality; the list is a potential portfolio and not a customer register. Corridors: A = Tequixquiac; B = Zumpango/Zitlaltepec; C = Metropolitan. Source: Own elaboration with company information, Guiamexican (n.d.) and MXFirmas (n.d.).

Appendix B. Python source code of the VRP model

The prototype consists of four modules that must be located in the same folder. Running it requires Python 3.10 or later with NumPy, SciPy and Matplotlib; experiments.py is run first, which generates results.json, and then plots.py, which produces the figures of the article.

B.1 Module data.py

```
"""
data.py - Input data of the last-mile distribution VRP model.
 Contains: (i) the 35 potential destinations from the diagnosis with their zone,
business type, one-way distance and time from the plant; (ii) approximate
reference coordinates; (iii) the reproducible generation of demands and order
release times; and (iv) the construction of the distance and time matrices.
"""
import numpy as np
 # ---------------------------------------------------------------------------
# General parameters (company diagnosis and engineering assumptions)
# ---------------------------------------------------------------------------
FUEL_PRICE = 23.70          # MXN/L, national reference Oct-2026
FUEL_ECONOMY = 7.0          # km/L, conservative assumption for a light truck
CAPACITIES = [600, 800]     # kg, vehicle 1 and vehicle 2
VEHICLE_NAMES = ["V1 (600 kg)", "V2 (800 kg)"]
HORIZON = 480               # min, working day 07:00-15:00
```

```
LOAD_TIME = 20              # min of loading/reloading at the plant per trip
SERV_RETAIL = 10            # min of service per stop at a retail store
SERV_LARGE = 15             # min per stop at a supermarket, club or wholesaler
LAMBDA_TRIP = 50.0          # equivalent MXN per activated trip (lambda 1)
LAMBDA_OVERTIME = 10.0      # equivalent MXN per minute beyond the working day (lambda 2)
WAVES = [0, 150, 270]       # production release, min after 07:00: 07:00, 09:30, 11:30
WAVE_PROB = [0.45, 0.35, 0.20]
DEPOT = ("Plant", 19.9750, -99.1700)   # center of Apaxco de Ocampo
# id, zone/corridor, business type, locality, one-way km, one-way min, lat, lon
DESTINATIONS = [
    (1, "A", "Dairy shop / mini-market", "Tequixquiac", 5, 10, 19.9135, -99.1480),
    (2, "A", "Grocery / mini-market", "Tequixquiac", 6, 12, 19.9075, -99.1395),
    (3, "A", "Grocery", "Tequixquiac", 6, 12, 19.9068, -99.1405),
    (4, "A", "Self-service store", "Tequixquiac", 5, 10, 19.9098, -99.1430),
    (5, "A", "Supermarket", "Tequixquiac", 5, 10, 19.9092, -99.1418),
    (6, "A", "Mini-market / grocery", "Tequixquiac", 5, 10, 19.9101, -99.1425),
    (7, "A", "Cheese shop / dairy", "Tequixquiac", 5, 10, 19.9095, -99.1435),
    (8, "A", "Mini-market", "Tequixquiac", 8, 15, 19.8985, -99.1300),
    (9, "A", "Producer / cheese shop", "Tequixquiac", 6, 12, 19.9040, -99.1460),
    (10, "B", "Dairy shop", "Zumpango", 21, 32, 19.8000, -99.1010),
    (11, "B", "Dairy shop", "Zumpango", 21, 32, 19.7945, -99.0965),
    (12, "B", "Dairy shop", "Zumpango", 21, 32, 19.7985, -99.0985),
    (13, "B", "Dairy shop", "Zumpango", 21, 32, 19.7990, -99.0970),
    (14, "B", "Dairy shop / butcher", "Zumpango", 22, 33, 19.8005, -99.0950),
    (15, "B", "Dairy shop", "Zumpango", 21, 32, 19.7930, -99.0990),
    (16, "B", "Dairy shop", "Zumpango", 21, 32, 19.7998, -99.0960),
    (17, "B", "Dairy shop / grocery", "Zumpango", 21, 32, 19.7955, -99.1025),
    (18, "B", "Dairy shop / dairy products", "Zumpango", 21, 32, 19.7930, -99.1040),
    (19, "B", "Dairy store", "San Miguel Bocanegra", 24, 35, 19.8150, -99.0700),
```

```
    (20, "B", "Dairy store", "San Juan Zitlaltepec", 15, 22, 19.8190, -99.1430),
    (21, "B", "Self-service store", "Zumpango (Buenavista)", 23, 33, 19.7700, -99.0560),
    (22, "C", "Warehouse club", "Zumpango (Buenavista)", 23, 33, 19.7690, -99.0580),
    (23, "C", "Self-service store", "Paseos de San Juan", 27, 38, 19.7580, -99.0460),
    (24, "C", "Self-service store", "Zumpango (Bo. Santiago)", 22, 32, 19.7920, -99.0930),
    (25, "C", "Wholesale / grocery", "Zumpango (Bo. San Lorenzo)", 22, 32, 19.7860, -99.0850),
    (26, "C", "Wholesale / grocery", "San Sebastián", 23, 34, 19.7800, -99.0700),
    (27, "C", "Self-service store", "Tecámac (Sierra Hermosa)", 37, 48, 19.7350, -98.9900),
    (28, "C", "Self-service store", "Tecámac (Hueyotenco)", 39, 50, 19.7020, -98.9800),
    (29, "C", "Supermarket", "Tecámac (Center)", 40, 50, 19.7130, -98.9690),
    (30, "C", "Wholesale market", "Tecámac (Ajoloapan)", 44, 55, 19.7600, -98.9500),
    (31, "C", "Warehouse club", "Tecámac", 39, 50, 19.7050, -98.9850),
    (32, "C", "Self-service store", "Ecatepec", 56, 70, 19.6200, -99.0050),
    (33, "C", "Hypermarket", "Ecatepec", 59, 72, 19.6030, -99.0430),
    (34, "C", "Dairy shop / wholesaler", "Ecatepec", 57, 70, 19.6420, -99.0270),
    (35, "C", "Wholesale market", "Tultitlán", 55, 68, 19.6460, -99.1530),
]
N = len(DESTINATIONS)
CORRIDORS = {"A": "Tequixquiac", "B": "Zumpango / Zitlaltepec", "C": "Metropolitan"}


LARGE = ("Self-service store", "Supermarket", "Warehouse club", "Hypermarket")
WHOLESALE = ("Wholesale", "wholesaler")
 def segment(business):
    """Classifies the business type: 0 retail, 1 large format, 2 wholesale."""
    if any(m in business for m in WHOLESALE):
        return 2
    if any(g in business for g in LARGE):
        return 1
    return 0
```

```
SEGMENT = np.array([segment(d[2]) for d in DESTINATIONS])
DEMAND_RANGE = {0: (20, 25), 1: (28, 34), 2: (38, 46)}   # kg per order
 def generate_demand(seed=2026, factor=1.0):
    """Demand per customer (kg): 20-25 kg for retail stores (company data)
    and larger orders for large-format and wholesale customers to reconcile
    the reported daily total (> 900 kg)."""
    rng = np.random.default_rng(seed)
    q = [rng.uniform(*DEMAND_RANGE[s]) for s in SEGMENT]
    return np.round(np.array(q) * factor).astype(int)
def generate_release(seed=2026):
    """Time (min after 07:00) at which each order is ready to be loaded."""
    rng = np.random.default_rng(seed + 1)
    return np.array(rng.choice(WAVES, size=N, p=WAVE_PROB))
def service_times(factor=1.0):
    s = np.where(SEGMENT == 0, SERV_RETAIL, SERV_LARGE) * factor
    return np.concatenate([[0], s]).round().astype(int)
def haversine(a, b):
    r = 6371.0
    la1, lo1, la2, lo2 = map(np.radians, (a[0], a[1], b[0], b[1]))
    h = (np.sin((la2 - la1) / 2) ** 2
        + np.cos(la1) * np.cos(la2) * np.sin((lo2 - lo1) / 2) ** 2)
    return 2 * r * np.arcsin(np.sqrt(h))
def coordinates():
    pts = [(DEPOT[1], DEPOT[2])] + [(d[6], d[7]) for d in DESTINATIONS]
    return np.array(pts)
def calibrate():
    """Circuity factor k (median of road km / geodesic km) and linear time
    model t = a + b*d fitted with the 35 data points from the diagnosis."""
    xy = coordinates()
```

```
    geo = np.array([haversine(xy[0], xy[i]) for i in range(1, N + 1)])
    km = np.array([d[4] for d in DESTINATIONS], float)
    mins = np.array([d[5] for d in DESTINATIONS], float)
    k = float(np.median(km / geo))
    b, a = np.polyfit(km, mins, 1)
    pred = a + b * km
    r2 = 1 - np.sum((mins - pred) ** 2) / np.sum((mins - mins.mean()) ** 2)
    return {"k": k, "a": float(a), "b": float(b), "r2": float(r2),
            "geo": geo, "km": km}
def matrices(k_fixed=None):
    """Origin-destination matrix of distance (km) and time (min) for 36 nodes.
    Plant-customer arcs: values from the diagnosis. Customer-customer arcs:
    geodesic distance x circuity factor and time from the linear model."""
    cal = calibrate()
    # A road distance cannot be shorter than the geodesic distance: k >= 1
    cal["k_used"] = k_fixed if k_fixed else max(1.0, cal["k"])
    xy = coordinates()
    n = N + 1
    D = np.zeros((n, n))
    T = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i == j:
                continue
            if i == 0 or j == 0:
                c = DESTINATIONS[max(i, j) - 1]
                D[i, j], T[i, j] = c[4], c[5]
            else:
                D[i, j] = max(0.5, cal["k_used"] * haversine(xy[i], xy[j]))
```

```
            T[i, j] = cal["a"] + cal["b"] * D[i, j]
    return np.round(D, 2), np.round(T, 1), cal
```

B.2 Module model.py

```
"""
model.py - Capacitated, heterogeneous, multi-trip VRP model with order
release times (MT-CVRP with replenishment).
 Components:
  evaluate()          objective function Z and time schedule of a solution
  savings()           Clarke and Wright constructive heuristic
  annealing()         local search by simulated annealing (metaheuristic)
  current_operation() reactive dispatching rules (baseline scenario)
  exact_tsp()         MILP verification (DFJ cuts) of the sequence of each trip
"""
import math
import random
import numpy as np
from scipy.optimize import milp, LinearConstraint, Bounds
import data as dt
 class Instance:
    def __init__(self, D, T, q, r, s, cap=dt.CAPACITIES, econ=dt.FUEL_ECONOMY,
                 price=dt.FUEL_PRICE, H=dt.HORIZON, load=dt.LOAD_TIME,
                 l1=dt.LAMBDA_TRIP, l2=dt.LAMBDA_OVERTIME):
        self.D, self.T = D, T
        self.q = np.concatenate([[0], q])       # demand, index 0 = plant
        self.r = np.concatenate([[0], r])       # order release time
        self.s = s                              # service time
        self.cap, self.econ, self.price = cap, econ, price
        self.H, self.load, self.l1, self.l2 = H, load, l1, l2
        self.n = len(q)
```

```
def evaluate(inst, sol, detail=False):
    """sol = [trips_V1, trips_V2]; each trip is a list of customers.
    Z = fuel cost + l1*trips + l2*minutes beyond the working day."""
    km = 0.0
    trips_n = 0
    overtime = 0.0
    excess = 0
    schedule = []
    end = []
    for v, trips in enumerate(sol):
        t = 0.0
        for trip in trips:
            if not trip:
                continue
            trips_n += 1
            load = sum(inst.q[i] for i in trip)
            excess += max(0, load - inst.cap[v])
            ready = max(inst.r[i] for i in trip)
            start = max(t, ready)              # wait for the release
            t = start + inst.load              # loading at the plant
            departure = t
            prev = 0
            stops = []
            for i in trip:
                km += inst.D[prev, i]
                t += inst.T[prev, i]
                stops.append((i, t))
                t += inst.s[i]
                prev = i
```

```
            km += inst.D[prev, 0]
            t += inst.T[prev, 0]
            schedule.append(dict(veh=v, start=start, departure=departure,
                                 ret=t, load=load, customers=trip[:],
                                 stops=stops))
        overtime += max(0.0, t - inst.H)
        end.append(t)
    liters = km / inst.econ
    cost = liters * inst.price
    Z = cost + inst.l1 * trips_n + inst.l2 * overtime + 1e4 * excess
    if not detail:
        return Z
    return dict(Z=Z, km=km, liters=liters, cost=cost, trips=trips_n,
                overtime=overtime, end=end, schedule=schedule, excess=excess)
# ---------------------------------------------------------------------------
# Constructive heuristic: Clarke and Wright savings (parallel version)
# ---------------------------------------------------------------------------
def savings(inst):
    n, D, q = inst.n, inst.D, inst.q
    capmax = max(inst.cap)
    routes = {i: [i] for i in range(1, n + 1)}
    of = {i: i for i in range(1, n + 1)}
    pairs = sorted(((D[i, 0] + D[0, j] - D[i, j], i, j)
                    for i in range(1, n + 1) for j in range(i + 1, n + 1)),
                   reverse=True)
    for s, i, j in pairs:
        ri, rj = of[i], of[j]
        if ri == rj:
            continue
```

```
        a, b = routes[ri], routes[rj]
        if sum(q[k] for k in a) + sum(q[k] for k in b) > capmax:
            continue
        if a[-1] == i and b[0] == j:
            new = a + b
        elif a[0] == i and b[-1] == j:
            new = b + a
        elif a[0] == i and b[0] == j:
            new = a[::-1] + b
        elif a[-1] == i and b[-1] == j:
            new = a + b[::-1]
        else:
            continue
        routes[ri] = new
        del routes[rj]
        for k in new:
            of[k] = ri
    return assign_vehicles(inst, list(routes.values()))
def assign_vehicles(inst, routes):
    """Assigns routes to vehicles (as trips) balancing the end time."""
    sol = [[], []]
    free = [0.0, 0.0]
    for route in sorted(routes, key=lambda r: max(inst.r[i] for i in r)):
        load = sum(inst.q[i] for i in route)
        candidates = [v for v in (0, 1) if load <= inst.cap[v]]
        v = min(candidates, key=lambda v: free[v])
        sol[v].append(route)
        free[v] = evaluate(inst, sol, True)["end"][v]
    return sol
```

```
# ---------------------------------------------------------------------------
# Metaheuristic: simulated annealing with routing neighborhoods
# ---------------------------------------------------------------------------
def _copy(sol):
    return [[t[:] for t in v] for v in sol]
def _clean(sol):
    return [[t for t in v if t] for v in sol]
def _neighbor(inst, sol, rng):
    s = _copy(sol)
    trips = [(v, k) for v in (0, 1) for k in range(len(s[v]))]
    m = rng.random()
    if m < 0.40:                          # relocate a customer
        v1, k1 = rng.choice(trips)
        i = s[v1][k1].pop(rng.randrange(len(s[v1][k1])))
        if rng.random() < 0.06:           # open a new trip
            v2 = rng.randrange(2)
            s[v2].insert(rng.randrange(len(s[v2]) + 1), [i])
        else:
            v2, k2 = rng.choice(trips)
            t2 = s[v2][k2]
            t2.insert(rng.randrange(len(t2) + 1), i)
    elif m < 0.60:                        # swap
        (v1, k1), (v2, k2) = rng.choice(trips), rng.choice(trips)
        a, b = s[v1][k1], s[v2][k2]
        x, y = rng.randrange(len(a)), rng.randrange(len(b))
        a[x], b[y] = b[y], a[x]
    elif m < 0.78:                        # intra-trip 2-opt
        v1, k1 = rng.choice(trips)
        t = s[v1][k1]       if len(t) > 2:
```

```
            x, y = sorted(rng.sample(range(len(t)), 2))
            t[x:y + 1] = t[x:y + 1][::-1]
        elif m < 0.90:                      # or-opt (segment)
            v1, k1 = rng.choice(trips)
            t = s[v1][k1]
            L = min(len(t), rng.choice((2, 3)))
            x = rng.randrange(len(t) - L + 1)
            seg = t[x:x + L]
            del t[x:x + L]
            v2, k2 = rng.choice(trips)
            t2 = s[v2][k2]
            p = rng.randrange(len(t2) + 1)
            t2[p:p] = seg
        else:                               # move/reorder a trip
            v1, k1 = rng.choice(trips)
            trip = s[v1].pop(k1)
            v2 = rng.randrange(2)
            s[v2].insert(rng.randrange(len(s[v2]) + 1), trip)
        s = _clean(s)
        for v in (0, 1):                    # capacity feasibility
            for t in s[v]:
                if sum(inst.q[i] for i in t) > inst.cap[v]:
                    return None
        return s
    def annealing(inst, initial, iters=60000, seed=1, T0=None, Tf=0.05,
                  log=False):
        rng = random.Random(seed)
        cur = _copy(initial)
        z_cur = evaluate(inst, cur)
```

```
    best, z_best = _copy(cur), z_cur
    T = T0 if T0 else 0.02 * z_cur
    alpha = (Tf / T) ** (1.0 / iters)
    hist = []
    for it in range(iters):
        new = _neighbor(inst, cur, rng)
        if new is not None:
            z = evaluate(inst, new)
            if z <= z_cur or rng.random() < math.exp((z_cur - z) / T):
                cur, z_cur = new, z
                if z < z_best - 1e-9:
                    best, z_best = _copy(new), z
        T *= alpha
        if log and it % 250 == 0:
            hist.append((it, z_cur, z_best))
    best = improve_sequences(inst, best)
    return (best, hist) if log else best
def improve_sequences(inst, sol):
    """Exhaustive 2-opt within each trip (final polishing)."""
    sol = _copy(sol)
    D = inst.D
    for v in sol:
        for t in v:
            improved = True
            while improved:
                improved = False
                route = [0] + t + [0]
                for a in range(1, len(route) - 2):
                    for b in range(a + 1, len(route) - 1):
```

```
                d0 = D[route[a - 1], route[a]] + D[route[b], route[b + 1]]
                d1 = D[route[a - 1], route[b]] + D[route[a], route[b + 1]]
                if d1 < d0 - 1e-9:
                    route[a:b + 1] = route[a:b + 1][::-1]
                    improved = True
            t[:] = route[1:-1]
    return sol
# ---------------------------------------------------------------------------
# Baseline scenario: reactive dispatching rules of the current operation
# ---------------------------------------------------------------------------
def current_operation(inst, zones):
    """Each vehicle leaves as soon as released orders exist, serves a single
    corridor per trip (main connection), visits customers in increasing order
    of distance from the plant and returns for more product."""
    pending = set(range(1, inst.n + 1))
    sol = [[], []]
    free = [0.0, 0.0]
    while pending:
        v = min((0, 1), key=lambda k: (free[k], -inst.cap[k]))
        t = free[v]
        avail = [i for i in pending if inst.r[i] <= t]
        if not avail:
            free[v] = min(inst.r[i] for i in pending)
            continue
        kg = {}
        for i in avail:
            kg[zones[i]] = kg.get(zones[i], 0) + inst.q[i]
        z = max(kg, key=kg.get)
        trip, load = [], 0
```

```
        for i in sorted([i for i in avail if zones[i] == z],
                        key=lambda i: inst.D[0, i]):
            if load + inst.q[i] <= inst.cap[v]:
                trip.append(i)
                load += inst.q[i]
        sol[v].append(trip)
        pending -= set(trip)
        free[v] = evaluate(inst, sol, True)["end"][v]
    return sol

# ---------------------------------------------------------------------------
# Exact verification of the sequence of each trip: symmetric TSP solved as a
# MILP (HiGHS) with iterative subtour elimination (DFJ cuts)
# ---------------------------------------------------------------------------
def exact_tsp(D, nodes, max_iter=200):
    V = [0] + list(nodes)
    m = len(V)
    if m <= 3:
        return sum(D[V[i], V[(i + 1) % m]] for i in range(m)), 0
    edges = [(a, b) for a in range(m) for b in range(a + 1, m)]
    c = np.array([(D[V[a], V[b]] + D[V[b], V[a]]) / 2 for a, b in edges])
    rows, lo, hi = [], [], []
    for a in range(m):                    # degree 2 at every node
        f = np.array([1.0 if a in e else 0.0 for e in edges])
        rows.append(f); lo.append(2); hi.append(2)
    cuts = 0
    for _ in range(max_iter):
        res = milp(c, constraints=LinearConstraint(np.array(rows), lo, hi),
                   integrality=np.ones(len(c)), bounds=Bounds(0, 1))
        x = np.round(res.x).astype(int)
```

```
        # connected components of the solution
        adj = {a: [] for a in range(m)}
        for k, (a, b) in enumerate(edges):
            if x[k]:
                adj[a].append(b); adj[b].append(a)
        seen, comps = set(), []
        for a in range(m):
            if a in seen:
                continue
            stack, comp = [a], set()
            while stack:
                u = stack.pop()
                if u in comp:
                    continue
                comp.add(u); stack.extend(adj[u])
            seen |= comp; comps.append(comp)
        if len(comps) == 1:
            return res.fun, cuts
        for S in comps:                    # cut: sum x(S) <= |S| - 1
            f = np.array([1.0 if (a in S and b in S) else 0.0 for a, b in edges])
            rows.append(f); lo.append(-np.inf); hi.append(len(S) - 1)
            cuts += 1
    return np.nan, cuts
def trip_km(D, trip):
    route = [0] + trip + [0]
    return sum(D[route[k], route[k + 1]] for k in range(len(route) - 1))
```

B.3 Module experiments.py

```
"""
experiments.py - Runs the scenarios, the exact verification, the sensitivity
```

```
analyses and the Monte Carlo simulation. Saves results.json.
"""
import json
import time
import numpy as np
import data as dt
import model as m
 SEEDS = range(1, 11)          # independent simulated annealing replications
ITERS = 300000                # iterations per run
ZONES = {i + 1: d[1] for i, d in enumerate(dt.DESTINATIONS)}
  def summary(inst, sol):
    e = m.evaluate(inst, sol, True)
    q_tot = inst.q.sum()
    return dict(
        km=e["km"], liters=e["liters"], cost=e["cost"], trips=e["trips"],
        overtime=e["overtime"], Z=e["Z"], end=e["end"],
        duration=sum(e["end"]),
        km_delivery=e["km"] / inst.n, cost_kg=e["cost"] / q_tot,
        cost_delivery=e["cost"] / inst.n,
        utilization=[a["load"] / inst.cap[a["veh"]] for a in e["schedule"]],
        schedule=[dict(veh=a["veh"], start=a["start"], departure=a["departure"],
                       ret=a["ret"], load=int(a["load"]),
                       customers=[int(c) for c in a["customers"]],
                       km=m.trip_km(inst.D, a["customers"]))
                  for a in e["schedule"]],
        sol=[[[int(c) for c in t] for t in v] for v in sol])
 def optimize(inst, seeds=SEEDS, iters=ITERS, log=False):
   ini = m.savings(inst)
   runs, best, z_best, hist_best, times = [], None, np.inf, None, []
```

```
    for s in seeds:
        t0 = time.time()
        if log:
            sol, hist = m.annealing(inst, ini, iters=iters, seed=s, log=True)
        else:
            sol, hist = m.annealing(inst, ini, iters=iters, seed=s), None
        times.append(time.time() - t0)
        z = m.evaluate(inst, sol)
        runs.append(z)
        if z < z_best:
            best, z_best, hist_best = sol, z, hist
    return best, ini, dict(Z=runs, t=times), hist_best
def instance(q=None, r=None, s=None, D=None, T=None, **kw):
    if D is None:
        D, T, _ = dt.matrices()
    q = dt.generate_demand() if q is None else q
    r = dt.generate_release() if r is None else r
    s = dt.service_times() if s is None else s
    return m.Instance(D, T, q, r, s, **kw)
def main():
    R = {}
    D, T, cal = dt.matrices()
    q, r, s = dt.generate_demand(), dt.generate_release(), dt.service_times()
    R["calibration"] = {k: cal[k] for k in ("k", "k_used", "a", "b", "r2")}
    R["demand"] = q.tolist(); R["release"] = r.tolist(); R["service"] = s.tolist()
    R["D"] = D.tolist(); R["T"] = T.tolist()
     inst_B = m.Instance(D, T, q, r, s)
    inst_A = m.Instance(D, T, q, np.zeros_like(r), s)
```

```
# Baseline scenario: current rules with staggered release
R["base"] = summary(inst_B, m.current_operation(inst_B, ZONES))
# Scenario A: static CVRP (everything ready at 07:00)
solA, iniA, runsA, histA = optimize(inst_A, log=True)
R["A"] = summary(inst_A, solA); R["A"]["runs"] = runsA
R["A"]["cw"] = summary(inst_A, iniA); R["A"]["hist"] = histA
# Scenario B: multiple trips with release times
solB, iniB, runsB, histB = optimize(inst_B, log=True)
R["B"] = summary(inst_B, solB); R["B"]["runs"] = runsB
R["B"]["cw"] = summary(inst_B, iniB); R["B"]["hist"] = histB
print("scenarios done", flush=True)
 # Exact verification of the sequence of each trip (MILP with DFJ cuts, HiGHS)
ver = []
for sc in ("A", "B"):
    for a in R[sc]["schedule"]:
        opt, cuts = m.exact_tsp(D, a["customers"])
        ver.append(dict(sc=sc, veh=a["veh"], n=len(a["customers"]),
                        heur=a["km"], exact=opt))
R["verification"] = ver
print("verification done", flush=True)
 # Sensitivity (2 seeds x 150,000 iterations per configuration)
sens = {"demand": [], "service": [], "circuity": [], "release": []}
for f in (1.0, 1.1, 1.2, 1.3, 1.4):
    ins = instance(q=dt.generate_demand(factor=f), D=D, T=T)
    b = summary(ins, m.current_operation(ins, ZONES))
    o = summary(ins, optimize(ins, (1, 2), 150000)[0])
    sens["demand"].append(dict(f=f, kg=int(ins.q.sum()), base=b["Z"], opt=o["Z"],
                               km_b=b["km"], km_o=o["km"], v_b=b["trips"],
                               v_o=o["trips"], x_b=b["overtime"], x_o=o["overtime"]))
```

```
for f in (0.8, 1.0, 1.2, 1.4):
    ins = instance(s=dt.service_times(f), D=D, T=T)
    b = summary(ins, m.current_operation(ins, ZONES))
    o = summary(ins, optimize(ins, (1, 2), 150000)[0])
    sens["service"].append(dict(f=f, base=b["Z"], opt=o["Z"], km_b=b["km"],
                                km_o=o["km"], x_b=b["overtime"], x_o=o["overtime"]))
for k in (1.0, 1.15, 1.30):
    Dk, Tk, _ = dt.matrices(k_fixed=k)
    ins = instance(D=Dk, T=Tk)
    b = summary(ins, m.current_operation(ins, ZONES))
    o = summary(ins, optimize(ins, (1, 2), 150000)[0])
    sens["circuity"].append(dict(k=k, base=b["Z"], opt=o["Z"], km_b=b["km"],
                                 km_o=o["km"], x_b=b["overtime"], x_o=o["overtime"]))
for w in ([0, 150, 270], [0, 120, 210], [0, 90, 150], [0, 60, 90], [0, 0, 0]):
    rr = np.array([w[dt.WAVES.index(x)] for x in r])
    ins = instance(r=rr, D=D, T=T)
    b = summary(ins, m.current_operation(ins, ZONES))
    o = summary(ins, optimize(ins, (1, 2), 150000)[0])
    sens["release"].append(dict(waves=w, base=b["Z"], opt=o["Z"],
                                km_b=b["km"], km_o=o["km"], v_b=b["trips"],
                                v_o=o["trips"], x_b=b["overtime"], x_o=o["overtime"],
                                c_b=b["cost"], c_o=o["cost"]))
R["sensitivity"] = sens
print("sensitivity done", flush=True)
 # Monte Carlo: 30 simulated days with random demand and release
mc = []
for rep in range(30):
    ins = instance(q=dt.generate_demand(100 + rep),
                   r=dt.generate_release(100 + rep), D=D, T=T)
```

```
        b = summary(ins, m.current_operation(ins, ZONES))
        o = summary(ins, optimize(ins, (1, 2), 120000)[0])
        mc.append(dict(kg=int(ins.q.sum()), km_b=b["km"], km_o=o["km"],
                       c_b=b["cost"], c_o=o["cost"], v_b=b["trips"],
                       v_o=o["trips"], x_b=b["overtime"], x_o=o["overtime"],
                       Z_b=b["Z"], Z_o=o["Z"]))
    R["montecarlo"] = mc
    with open("results.json", "w") as f:
        json.dump(R, f, default=float, indent=1)
    print("done", flush=True)
if __name__ == "__main__":
    main()
```

B.4 Module plots.py

```
"""
plots.py - Figures of the article generated from results.json.
"""
import json
import numpy as np
import matplotlib
import matplotlib.ticker
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from matplotlib.patches import FancyBboxPatch, FancyArrowPatch
from matplotlib.lines import Line2D
import data as dt
plt.rcParams.update({
    "font.family": "DejaVu Serif", "font.size": 9, "axes.titlesize": 10,
    "axes.edgecolor": "#8a8984", "axes.labelcolor": "#2b2b2a",
    "xtick.color": "#52514e", "ytick.color": "#52514e",
```

```
    "axes.spines.top": False, "axes.spines.right": False,
    "grid.color": "#e4e3df", "grid.linewidth": 0.6,
})
INK, INK2 = "#0b0b0b", "#52514e"
C_CORR = {"A": "#1baf7a", "B": "#eda100", "C": "#6250d6"}
C_VEH = ["#2a78d6", "#eb6834"]
C_SC = {"base": "#8a8984", "A": "#2a78d6", "B": "#eb6834"}
N_SC = {"base": "Current operation", "A": "Scenario A (CVRP)",
        "B": "Scenario B (MT-CVRP)"}
DPI = 220
R = json.load(open("results.json"))
XY = dt.coordinates()
ZONE = [None] + [d[1] for d in dt.DESTINATIONS]
 def save(fig, name):
    fig.savefig(f"figs/{name}.png", dpi=DPI, bbox_inches="tight",
                facecolor="white")
    plt.close(fig)
def clock(m):
    return f"{7 + int(m // 60):02d}:{int(m % 60):02d}"
# ---------------------------------------------------------------- maps
LOCALITIES = [("Apaxco", 19.9750, -99.1700, (6, 6)),
              ("Tequixquiac", 19.9095, -99.1430, (-70, -4)),
              ("Zumpango", 19.7969, -99.0997, (-58, 8)),
              ("Zitlaltepec", 19.8190, -99.1430, (-62, 2)),
              ("Tecámac", 19.7130, -98.9690, (8, -10)),
              ("Ecatepec", 19.6100, -99.0250, (8, -14)),
              ("Tultitlán", 19.6460, -99.1530, (-52, -12))]
  def base_map(ax, title=None):
    ax.set_aspect(1 / np.cos(np.radians(19.8)))
```

```
    ax.set_xlim(-99.20, -98.92)
    ax.set_ylim(19.58, 20.00)
    ax.grid(True, zorder=0)
    ax.set_xlabel("Longitude (°)")
    ax.set_ylabel("Latitude (°)")
    ax.tick_params(labelsize=7.5)
    for name, la, lo, off in LOCALITIES:
        ax.annotate(name, (lo, la), xytext=off, textcoords="offset points",
                    fontsize=7.5, color=INK2, style="italic")
    # 10 km scale bar and north arrow
    x0, y0 = -99.19, 19.618
    dlon = 10 / (111.32 * np.cos(np.radians(19.6)))
    ax.plot([x0, x0 + dlon], [y0, y0], color=INK, lw=2)
    ax.text(x0 + dlon / 2, y0 + 0.006, "10 km", ha="center", fontsize=7.5)
    ax.annotate("N", xy=(-98.94, 19.875), xytext=(-98.94, 19.845),
                ha="center", fontsize=9, weight="bold",
                arrowprops=dict(arrowstyle="-|>", color=INK, lw=1.2))
    ax.plot(XY[0, 1], XY[0, 0], marker="s", ms=9, color=INK, zorder=6)
    if title:
        ax.set_title(title, loc="left")
def fig_area():
    q = np.array(R["demand"])
    fig, ax = plt.subplots(figsize=(6.6, 7.4))
    base_map(ax)
    for z in "ABC":
        idx = [i for i in range(1, dt.N + 1) if ZONE[i] == z]
        ax.scatter(XY[idx, 1], XY[idx, 0], s=q[np.array(idx) - 1] * 2.2,
                   color=C_CORR[z], edgecolor="white", lw=0.8, zorder=5,
                   label=f"Corridor {z}: {dt.CORRIDORS[z]} ({len(idx)})")
```

```
    ax.scatter([], [], marker="s", s=40, color=INK, label="Plant (depot)")
    for i in range(1, dt.N + 1):
        if i in (1, 4, 8, 10, 19, 20, 21, 23, 26, 27, 30, 32, 33, 34, 35):
            ax.annotate(str(i), (XY[i, 1], XY[i, 0]), xytext=(4, 3),
                        textcoords="offset points", fontsize=6.5, color=INK)
    ax.legend(loc="upper right", fontsize=7.5, frameon=True, framealpha=0.95,
              title="Potential destinations", title_fontsize=8)
    save(fig, "fig02_study_area")
def fig_routes(sc, name, title):
    fig, ax = plt.subplots(figsize=(6.6, 7.4))
    base_map(ax)
    styles = ["-", "--", ":", "-."]
    count = [0, 0]
    legend = []
    for a in R[sc]["schedule"]:
        v = a["veh"]
        ls = styles[count[v] % 4]
        count[v] += 1
        route = [0] + a["customers"] + [0]
        ax.plot(XY[route, 1], XY[route, 0], ls=ls, color=C_VEH[v], lw=1.6,
                alpha=0.9, zorder=4)
        legend.append(Line2D([0], [0], color=C_VEH[v], ls=ls, lw=1.8,
                             label=f"{dt.VEHICLE_NAMES[v]} · trip {count[v]} "
                                   f"({clock(a['departure'])}, {a['load']} kg)"))
    for z in "ABC":
        idx = [i for i in range(1, dt.N + 1) if ZONE[i] == z]
        ax.scatter(XY[idx, 1], XY[idx, 0], s=22, color=C_CORR[z],
                   edgecolor="white", lw=0.6, zorder=5)
    ax.legend(handles=legend, loc="upper right", fontsize=7, framealpha=0.95)
```

```
    r = R[sc]
    ax.text(-99.195, 19.59, f"{r['km']:.1f} km · {r['liters']:.1f} L · "
            f"${r['cost']:,.0f} MXN · {r['trips']} trips",
            fontsize=7.5, color=INK, bbox=dict(fc="white", ec="#c9c8c3"))
    save(fig, name)
# ---------------------------------------------------------- diagnosis
def fig_isolated_cost():
    km = np.array([d[4] for d in dt.DESTINATIONS])
    cost = 2 * km / dt.FUEL_ECONOMY * dt.FUEL_PRICE
    order = np.argsort(cost)
    fig, ax = plt.subplots(figsize=(7.2, 3.4))
    x = np.arange(dt.N)
    cols = [C_CORR[dt.DESTINATIONS[i][1]] for i in order]
    ax.bar(x, cost[order], color=cols, width=0.78, zorder=3)
    ax.set_xticks(x, [str(dt.DESTINATIONS[i][0]) for i in order], fontsize=6.5)
    ax.set_xlabel("Destination (identifier)")
    ax.set_ylabel("Fuel cost of the isolated\nround trip (MXN)")
    ax.grid(axis="y", zorder=0)
    hs = [plt.Rectangle((0, 0), 1, 1, color=C_CORR[z]) for z in "ABC"]
    ax.legend(hs, [f"Corridor {z}: {dt.CORRIDORS[z]}" for z in "ABC"],
              fontsize=7.5, frameon=False, loc="upper left")
    for i, j in enumerate(order):
        if i in (0, 9, 20, dt.N - 1):
            ax.text(i, cost[j] + 6, f"${cost[j]:.0f}", ha="center", fontsize=6.5)
    save(fig, "fig03_isolated_cost")
# ---------------------------------------------------------- diagrams
def box(ax, x, y, w, h, title, text, color):
    ax.add_patch(FancyBboxPatch((x, y), w, h, boxstyle="round,pad=0.01,rounding_size=0.02",
                                fc=color, ec="none", alpha=0.16))
```

```
    ax.add_patch(FancyBboxPatch((x, y), w, h, boxstyle="round,pad=0.01,rounding_size=0.02",
                                fc="none", ec=color, lw=1.2))
    ax.text(x + w / 2, y + h - 0.035, title, ha="center", va="top",
            weight="bold", fontsize=8.5, color=INK)
    ax.text(x + w / 2, y + h - 0.12, text, ha="center", va="top",
            fontsize=7, color=INK2, linespacing=1.35)
def arrow(ax, a, b, **kw):
    ax.add_patch(FancyArrowPatch(a, b, arrowstyle="-|>", mutation_scale=11,
                                 color=kw.get("c", "#52514e"), lw=1.1,
                                 connectionstyle=kw.get("cs", "arc3")))
def fig_design_thinking():
    fig, ax = plt.subplots(figsize=(7.6, 2.9))
    ax.set_xlim(0, 1); ax.set_ylim(0, 1); ax.axis("off")
    phases = [("1. Empathize", "Company information,\ncurrent operation,\n35 destinations and\nisolated costs"),
              ("2. Define", "Problem statement,\nrequirements and\nbaseline\nindicators"),
              ("3. Ideate", "Alternatives: CVRP,\nCVRP-TW and multi-\ntrip VRP with\norder release"),
              ("4. Prototype", "Python model:\ndata, matrices,\nsavings, simulated\nannealing and MILP"),
              ("5. Test", "Scenarios, 10\nreplications,\nsensitivity and\nMonte Carlo")]
    cols = ["#2a78d6", "#1baf7a", "#eda100", "#eb6834", "#6250d6"]
    w, h, y = 0.172, 0.50, 0.36
    xs = [0.01 + k * 0.199 for k in range(5)]
    for k, (t, s) in enumerate(phases):
        box(ax, xs[k], y, w, h, t, s, cols[k])
        if k < 4:
            arrow(ax, (xs[k] + w + 0.004, y + h / 2), (xs[k + 1] - 0.004, y + h / 2))
    arrow(ax, (xs[4] + w / 2, y - 0.02), (xs[1] + w / 2, y - 0.02),
          cs="arc3,rad=-0.25", c="#e34948")
    ax.text(0.5, 0.03, "Iteration: test results adjust the assumptions, "
```

```
        "requirements and scenarios of the model",
        ha="center", fontsize=7.5, color="#e34948", style="italic")
    save(fig, "fig01_design_thinking")
 def fig_architecture():
    fig, ax = plt.subplots(figsize=(7.6, 2.7))
    ax.set_xlim(0, 1); ax.set_ylim(0, 1); ax.axis("off")
    mods = [("data.py", "35 destinations,\ncoordinates, demand,\nrelease, matrices", 0.01, "#2a78d6"),
        ("model.py", "Objective function Z,\nsavings, simulated\nannealing, base rules", 0.27, "#eb6834"),
        ("experiments.py", "Scenarios, 10 seeds,\nMILP verification,\nsensitivity and\nMonte Carlo", 0.53,
"#1baf7a"),
        ("plots.py", "Route maps,\nGantt, indicators\nand distributions", 0.79, "#6250d6")]
    for t, s, x, c in mods:
      box(ax, x, 0.33, 0.2, 0.62, t, s, c)
    for x in (0.21, 0.47, 0.73):
      arrow(ax, (x + 0.005, 0.64), (x + 0.055, 0.64))
    ax.text(0.5, 0.16, "Libraries: NumPy · SciPy (milp/HiGHS) · Matplotlib · "
        "standard modules random and json", ha="center", fontsize=8, color=INK2)
    ax.text(0.5, 0.03, "Output: results.json → tables and figures of the article",
        ha="center", fontsize=8, color=INK2, style="italic")
    save(fig, "fig04_architecture")
 # ---------------------------------------------------- results
def fig_kpi():
    scs = ["base", "A", "B"]
    met = [("km", "Distance (km)", "{:.1f}"), ("cost", "Fuel (MXN)", "${:,.0f}"),
        ("trips", "Trips from the plant", "{:.0f}"), ("overtime", "Minutes beyond working day", "{:.0f}")]
    fig, axs = plt.subplots(1, 4, figsize=(7.6, 2.8))
    for ax, (k, t, f) in zip(axs, met):
      vals = [R[e][k] for e in scs]
      ax.bar(range(3), vals, color=[C_SC[e] for e in scs], width=0.7, zorder=3)
```

```
    for i, v in enumerate(vals):
        ax.text(i, v, f.format(v), ha="center", va="bottom", fontsize=7)
    ax.set_xticks(range(3), ["Current", "A", "B"])
    ax.set_title(t, fontsize=8.5)
    ax.grid(axis="y", zorder=0)
    ax.set_ylim(0, max(vals) * 1.18 + 1)
  fig.tight_layout()
  save(fig, "fig08_indicators")
def fig_gantt():
  scs = ["base", "A", "B"]
  fig, ax = plt.subplots(figsize=(7.6, 3.4))
  y = 0
  ticks, labs = [], []
  for e in scs:
    for v in (1, 0):
      for a in [a for a in R[e]["schedule"] if a["veh"] == v]:
        ax.barh(y, a["departure"] - a["start"], left=a["start"], height=0.6,
                color="#c9c8c3", zorder=3)
        ax.barh(y, a["ret"] - a["departure"], left=a["departure"], height=0.6,
                color=C_VEH[v], zorder=3, edgecolor="white", lw=1)
        ax.text((a["departure"] + a["ret"]) / 2, y, f"{a['load']} kg",
                ha="center", va="center", fontsize=6.3, color="white", zorder=4)
      ticks.append(y); labs.append(f"{['Current', 'A', 'B'][scs.index(e)]} · V{v + 1}")
      y += 1
    y += 0.5
  ax.axvline(dt.HORIZON, color="#e34948", lw=1.2, ls="--", zorder=2)
  ax.text(dt.HORIZON + 4, -0.25, "End of working day (15:00)", fontsize=7, color="#e34948")
  for o in dt.WAVES[1:]:
    ax.axvline(o, color="#8a8984", lw=0.8, ls=":", zorder=2)
```

```
    ax.set_yticks(ticks, labs, fontsize=7.5)
    xt = np.arange(0, 721, 60)
    ax.set_xticks(xt, [clock(t) for t in xt], fontsize=7.5)
    ax.set_xlim(0, 720)
    ax.invert_yaxis()
    ax.grid(axis="x", zorder=0)
    hs = [plt.Rectangle((0, 0), 1, 1, color=c) for c in ("#c9c8c3", C_VEH[0], C_VEH[1])]
    ax.legend(hs, ["Waiting and loading at the plant", "Trip V1 (600 kg)", "Trip V2 (800 kg)"],
              fontsize=7, loc="upper center", bbox_to_anchor=(0.5, 1.12), ncol=3,
              frameon=False)
    save(fig, "fig09_gantt")
def fig_utilization():
    fig, ax = plt.subplots(figsize=(7.2, 2.9))
    x = 0
    for e in ("base", "A", "B"):
        xs = []
        for a in R[e]["schedule"]:
            u = 100 * a["load"] / dt.CAPACITIES[a["veh"]]
            ax.bar(x, u, color=C_SC[e], width=0.75, zorder=3,
                   hatch="///" if a["veh"] == 1 else None, edgecolor="white")
            ax.text(x, u + 1.5, f"{u:.0f}%", ha="center", fontsize=6.5)
            xs.append(x); x += 1
        ax.text(np.mean(xs), -14, N_SC[e], ha="center", fontsize=7.5)
        x += 0.8
    ax.set_xticks([])
    ax.set_ylim(0, 105)
    ax.set_ylabel("Capacity utilization (%)")
    ax.grid(axis="y", zorder=0)
    hs = [plt.Rectangle((0, 0), 1, 1, fc="#bdbcb7"),
```

```
        plt.Rectangle((0, 0), 1, 1, fc="#bdbcb7", hatch="///", ec="white")]
    ax.legend(hs, ["Trip of V1 (600 kg)", "Trip of V2 (800 kg)"], fontsize=7,
              frameon=False, loc="upper right")
    save(fig, "fig10_utilization")
def fig_convergence():
    fig, axs = plt.subplots(1, 2, figsize=(7.6, 2.8))
    for ax, e in zip(axs, ("A", "B")):
        h = np.array(R[e]["hist"])
        ax.plot(h[:, 0] / 1000, h[:, 1], color="#bdbcb7", lw=0.8, label="Current solution")
        ax.plot(h[:, 0] / 1000, h[:, 2], color=C_SC[e], lw=2, label="Best solution")
        ax.axhline(R[e]["cw"]["Z"], color=INK2, ls="--", lw=1)
        ax.text(h[-1, 0] / 1000, R[e]["cw"]["Z"], "Savings (C-W)", ha="right",
                va="bottom", fontsize=7, color=INK2)
        ax.set_title(N_SC[e], fontsize=9)
        ax.set_xlabel("Iterations (thousands)")
        ax.set_ylabel("Z (equivalent MXN)")
        ax.grid(True)
        ax.legend(fontsize=7, frameon=False)
    fig.tight_layout()
    save(fig, "fig11_convergence")
def fig_sensitivity():
    S = R["sensitivity"]
    fig, axs = plt.subplots(1, 2, figsize=(7.6, 3.0))
    ax = axs[0]
    d = S["service"]
    x = [dt.SERV_RETAIL * r["f"] for r in d]
    ax.plot(x, [r["base"] for r in d], marker="o", color=C_SC["base"], lw=2, label="Current operation")
    ax.plot(x, [r["opt"] for r in d], marker="o", color=C_SC["B"], lw=2, label="Scenario B")
    ax.set_xticks(x, [f"{v:.0f} / {v * 1.5:.0f}" for v in x])
```

```
    ax.set_xlabel("Service per stop: retail / large format (min)")
    ax.set_ylabel("Z (equivalent MXN)")
    ax.set_title("a) Service time", loc="left", fontsize=9)
    ax.grid(True); ax.legend(fontsize=7, frameon=False)
    ax = axs[1]
    L = S["release"]
    lab = ["11:30", "10:30", "09:30", "08:30", "07:00"]
    ax.plot(range(len(L)), [r["base"] for r in L], marker="o", color=C_SC["base"], lw=2, label="Current operation")
    ax.plot(range(len(L)), [r["opt"] for r in L], marker="o", color=C_SC["B"], lw=2, label="Optimized scenario")
    ax.set_xticks(range(len(L)), lab)
    ax.set_xlabel("Release time of the last wave")
    ax.set_title("b) Production-dispatch synchronization", loc="left", fontsize=9)
    ax.grid(True); ax.legend(fontsize=7, frameon=False)
    fig.tight_layout()
    save(fig, "fig12_sensitivity")
def fig_montecarlo():
    mc = R["montecarlo"]
    red_km = [100 * (1 - r["km_o"] / r["km_b"]) for r in mc]
    red_z = [100 * (1 - r["Z_o"] / r["Z_b"]) for r in mc]
    dv = [r["v_b"] - r["v_o"] for r in mc]
    fig, axs = plt.subplots(1, 2, figsize=(7.6, 2.9), gridspec_kw={"width_ratios": [2, 1.2]})
    ax = axs[0]
    bp = ax.boxplot([red_km, red_z], orientation="horizontal", widths=0.5, patch_artist=True,
                    medianprops=dict(color=INK, lw=1.5))
    for p, c in zip(bp["boxes"], (C_SC["B"], "#6250d6")):
        p.set_facecolor(c); p.set_alpha(0.35); p.set_edgecolor(c)
    rng = np.random.default_rng(0)
    for k, vals in enumerate((red_km, red_z), start=1):        ax.scatter(vals, k + rng.uniform(-0.12, 0.12,
```

```
len(vals)), s=10, color=INK2, zorder=3)
    ax.set_yticks([1, 2], ["Distance\nreduction", "Reduction in Z\n(total cost)"])
    ax.set_xlabel("Reduction relative to the current operation (%)")
    ax.grid(axis="x")
    ax = axs[1]
    vals, cnt = np.unique(dv, return_counts=True)
    ax.bar(vals, cnt, color=C_SC["B"], width=0.6, zorder=3)
    ax.set_xlabel("Trips avoided per day")
    ax.set_ylabel("Simulated days")
    ax.set_xticks(vals)
    ax.yaxis.set_major_locator(matplotlib.ticker.MaxNLocator(integer=True))
    ax.grid(axis="y", zorder=0)
    fig.tight_layout()
    save(fig, "fig13_montecarlo")
if __name__ == "__main__":
    import os
    os.makedirs("figs", exist_ok=True)
    fig_design_thinking(); fig_area(); fig_isolated_cost(); fig_architecture()
    fig_routes("base", "fig05_routes_current", "Current operation")
    fig_routes("A", "fig06_routes_A", "Scenario A")
    fig_routes("B", "fig07_routes_B", "Scenario B")
    fig_kpi(); fig_gantt(); fig_utilization(); fig_convergence()
    fig_sensitivity(); fig_montecarlo()
    print("figures done")
```

Appendix C. Minimum data capture format for the pilot test

Table C1. Distribution log by stop or trip event

Date	Vehicle	Order / custom	Zone	Kg	Ready time	Plant departure	Arrival	Service start / end	Plant return	Odometer start /	Liters / cost	Reason for return

		er								**end**		

Note: When a vehicle returns to the plant for goods, the return is recorded as a separate event, with the km traveled, the time spent, the additional load and the orders that prompted the second departure. Source: Own elaboration.