Exercise Report: Units 6–10

Joan Fernández Navarro

HEURISTIC ALGORITHMS
IN TRANSPORT AND FINANCE

Master's Degree in Computational Engineering and Industrial Mathematics

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

The code developed to complete all the proposed exercises for this report is available in the online GitHub repository: https://github.com/JoanFer030/heuristic_exercises

1 Unit 6

1.1 Biased-Randomization for ARPO

Kizys et al., 2019 propose A Biased-Randomized Iterated Local Search Algorithm for Rich Portfolio Optimization (ARPO), a matheuristic designed to solve constrained mean–variance portfolio optimization problems with realistic features such as cardinality and quantity constraints. The problem is NP-hard due to these constraints, making exact optimization methods impractical for large instances. ARPO combines an Iterated Local Search (ILS) framework with quadratic programming and biased randomization. The algorithm first constructs a feasible initial portfolio based on assets with the highest individual returns, ensuring feasibility of the required return. It then iteratively improves the solution through perturbation (destruction–reconstruction using covariance-based "friendship" and biased randomization), local search via quadratic programming to optimize asset weights, and a credit-based acceptance criterion that allows limited solution degradation to escape local minima.
From a numerical perspective, ARPO demonstrates clear improvements over existing state-of-the-art approaches. When tested on the five standard ORlib benchmarks (Hang Seng, DAX 100, FTSE 100, S&P 100, and Nikkei 225), ARPO achieves very low average percentage loss (APL) relative to the unconstrained efficient frontier, ranging from as low as 0.00399% for Hang Seng and 0.20189% for Nikkei 225, to 1.88% for FTSE 100 and 4.65% for S&P 100. In addition, ARPO's robustness is confirmed through experiments with uncertain inputs. When random noise is added to expected returns, APL increases to around 3.3% on average, reflecting the higher cost of achieving target returns under uncertainty. By contrast, introducing randomness only in variances and covariances leads to negligible changes in performance.
Overall, these numerical results establish ARPO as a simple yet highly effective algorithm that consistently improves solution quality relative to existing approaches, while achieving these gains with lower computational effort and strong robustness to uncertainty and constraint variations. ARPO is robust to uncertainty in returns and stable under small variations in cardinality and quantity constraints, making it a simple, efficient, and flexible approach for rich portfolio optimization problems.

Score: 7

1.2 BR-DES Algorithm for Automated Storage and Retrieval Systems

Neroni et al., 2024 propose a biased-randomized discrete-event simulation optimization algorithm (BR-DES) to minimize the makespan in a realistic automated storage and retrieval system (AS/RS) tailored to the steel industry. The system under study is a

shuttle–lift–crane AS/RS (SLC-AS/RS), characterized by the handling of large, heavy, and non-standardized items subject to stringent weight and quality constraints. Due to the complex time dependencies, high resource interaction, and operational constraints inherent in such systems, the resulting scheduling and assignment problem is highly complex and unsuitable for exact optimization methods.

The proposed approach integrates a biased-randomized heuristic with discrete-event simulation (DES) within a multi-start framework. The DES component accurately captures machine synchronization, parallelism, and time dependencies among shuttles, lifts, and cranes, while the biased-randomized component introduces controlled randomness into key decision-making processes, including the sequencing of input/output requests, the allocation of storage locations, and the selection of bundles to fulfill customer orders. Biased randomization is implemented using geometric probability distributions, allowing the algorithm to explore near-greedy solutions efficiently while avoiding premature convergence. The algorithm generates multiple feasible solutions in short computational times, retaining only the best-performing ones.

From a numerical perspective, the BR-DES algorithm consistently outperforms alternative approaches across a wide range of test instances. Computational experiments are conducted on a real-world SLC-AS/RS layout with three racks (500 storage locations each), two input points, and two output points, using request lists ranging from 30 to 150 operations. Compared with a greedy heuristic, a random heuristic, and a tailored simulated annealing (SA) algorithm, BR-DES achieves the lowest makespan in all tested instances, while also being the only method that consistently finds feasible solutions across all scenarios.

But BR-DES not only reduces the makespan by a substantial margin compared to SA, while requiring only a few seconds of computation time, it also achieves markedly lower mismatches between required and retrieved quantities and quality levels, yielding solutions that are closer to customer specifications. The BR-DES obtains an average mismatch of 3,808 that is almost two times lower that the average mismatch of SA (5,109). Moreover, while SA exhibits high variability due to its exploration of infeasible solutions, BR-DES maintains stable performance thanks to its event-driven feasibility checks embedded in the simulation.

Overall, BR-DES delivers superior solution quality with lower computational effort and reduced variability, demonstrating its effectiveness in managing complex, time-dependent AS/RS operations in the steel industry.

Score: 10

1.3 A simheuristic approach for the 2L-VRP

Guimarans et al., 2018 propose a simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times (2L-SVRP), a realistic extension of the classical 2L-VRP in which customer demands consist of sets of non-stackable rectangular items and travel times are subject to uncertainty. The problem integrates routing and two-dimensional packing constraints under stochastic travel times and includes penalty costs associated with route overtime, resulting in a non-smooth stochastic op-

timisation problem that is computationally intractable for exact methods on medium and large instances.
To address this challenge, the authors develop a hybrid simheuristic algorithm that combines an Iterated Local Search (ILS) framework with Monte Carlo simulation and biased-randomised routing and packing heuristics. The constructive phase relies on a biased-randomised version of the Clarke–Wright savings heuristic for routing and a biased-randomised Best-Fit heuristic for two-dimensional packing, ensuring route feasibility by explicitly incorporating loading constraints during solution construction. During the improvement phase, the ILS applies controlled destruction–reconstruction mechanisms, 2-opt intra-route local search, and a simulated annealing–type acceptance criterion to balance intensification and diversification. Monte Carlo simulation is embedded at different stages of the search to evaluate candidate solutions under stochastic travel times and to guide the selection of elite solutions based on expected total cost, including overtime penalties.
From a numerical perspective, the proposed simheuristic clearly outperforms deterministic approaches when solutions are evaluated under stochastic conditions. Computational experiments are conducted on an extensive benchmark derived from the classical 2L-VRP instances, covering four instance classes (Classes 2–5) and up to 256 customers, with stochastic travel times modelled using Log-Normal distributions. Across all classes, solutions obtained by the simheuristic exhibit systematically lower expected costs than deterministic best-known solutions when both are evaluated in stochastic environments. On average, deterministic solutions suffer cost increases of around 2–3% when exposed to uncertainty, while the simheuristic reduces expected stochastic costs by approximately 2–4%, with improvements exceeding 7–10% in several medium and large instances. Additionally, the simheuristic significantly mitigates overtime penalties, achieving average reductions of 15–20 time units per instance relative to deterministic solutions.
Importantly, these gains are achieved with moderate computational effort: average runtimes remain within a few minutes even for the largest instances, making the approach suitable for practical applications. Overall, the numerical results demonstrate that solutions optimised for deterministic 2L-VRP instances can become highly suboptimal in stochastic settings, while the proposed simheuristic consistently produces more robust routing and packing plans. The study establishes simheuristics as an effective and flexible methodology for complex stochastic vehicle routing problems with integrated loading constraints.

Score: 8

1.4 A biased-randomized algorithm for the 2L-VRP

Dominguez et al., 2014 propose an efficient heuristic algorithm for the two-dimensional loading capacitated vehicle routing problem (2L-CVRP), a complex extension of the classical CVRP in which customer demands consist of multiple non-stackable rectangular items that must be packed into vehicles without overlap. The problem integrates routing decisions with two-dimensional packing constraints and considers realistic load-

ing settings, including unrestricted loading and the possibility of rotating items by 90 degrees.
To tackle this computationally challenging problem, the authors develop a multistart biased-randomized heuristic with a reduced number of robust parameters and a high degree of parallelizability. The approach integrates routing and packing decisions within a single constructive framework. At each multistart iteration, a biased-randomized version of the Clarke–Wright savings heuristic is used to generate candidate routes, where edge selection follows a geometric probability distribution that preserves the logic of the classical heuristic while promoting diversification. Route merging is accepted only if both vehicle capacity and two-dimensional packing feasibility are satisfied. Packing feasibility is verified using a biased-randomized enhanced Best-Fit heuristic that allows item rotations when solving the nonoriented loading variant. The algorithm is further strengthened by memory-based local search and a splitting-based improvement procedure, which decomposes promising solutions into smaller subproblems that are re-optimized independently before being recombined.
From a numerical perspective, the proposed method demonstrates strong performance on standard 2L-CVRP benchmark instances across all five item classes. In the nonoriented loading case, the algorithm consistently matches or improves upon the best-known solutions. On average, the proposed approach achieves negative gaps ranging from approximately 0.2% to 1.1% in BEST10 and AVG10 solution quality, with larger improvements observed in the more complex classes (Classes 2–4). In several instances, cost reductions exceed 2–4%, and in some large instances improvements above 7–9% are reported. These gains are obtained with significantly lower computational effort, as each replica is limited to 500 seconds, compared to up to three hours per replica in the reference ACO approach.
Score: 7.5

1.4.1 Implementation

To address the Two-Level Vehicle Routing Problem (2L-VRP), we implemented a solution based on the Clarke and Wright Savings heuristic extended with an explicit two-dimensional loading feasibility check. The overall implementation integrates routing decisions at the first level with a packing validation procedure at the second level, ensuring that each constructed route is not only feasible in terms of distance and weight, but also respects the 2D loading constraints of the vehicle.
The problem is modeled through a set of structured data classes representing the main entities of the 2L-VRP. Clients are defined by their spatial coordinates and a collection of rectangular items, each characterized by width, height, weight, and an associated client identifier. Vehicles are described by maximum width, height, and weight capacities, while an instance aggregates the depot, the set of clients, the vehicle specifications, and the rotation policy for items. This modular representation facilitates a clear separation between routing logic and packing feasibility checks.
The second level of the problem, corresponding to the loading constraints, is handled through a shelf-based packing heuristic. Items assigned to a route are sorted by decreasing maximum dimension and placed sequentially into horizontal shelves within the

vehicle. For each item, the algorithm attempts to place it into an existing shelf while minimizing remaining width; if this is not possible, a new shelf is created provided that the vehicle height constraint is not violated. When rotation is allowed, both orientations of each item are considered during placement. If any item cannot be placed within the vehicle dimensions, the packing solution is declared infeasible. This approach provides a fast and deterministic feasibility check that can be repeatedly invoked during route construction.

On the routing side, we implemented a biased randomized version of the Clarke and Wright Savings heuristic. Initially, each client is assigned to an individual route. Savings values are computed for every pair of clients based on the classical Clarke and Wright formulation, using Euclidean distances between clients and the depot. These savings are then sorted in descending order, but instead of selecting them deterministically, a biased random selection mechanism is applied. This mechanism assigns higher probabilities to larger savings while still allowing diversification, controlled by a parameter .

During the merging process, candidate routes corresponding to the selected saving are combined using all possible concatenation and reversal configurations. Each candidate merge is evaluated through a feasibility check that verifies both the total vehicle weight and the 2D packing constraints using the shelf packing heuristic. Only merges that satisfy all constraints are accepted. This ensures that the resulting routes remain feasible with respect to both levels of the problem throughout the construction process.

Listing 1: Python implementation of the Clarke and Wright Savings heuristic for the 2L-VRP, including biased randomization and shelf-based 2D packing constraints.

```
class ClarkWrightBRA:
    ...
    def solve(self) -> list[list[int]]:
        self.routes = [[i] for i in range(len(self.instance.clients))]
        self.client_to_route = {i: i for i in range(len(self.instance.clients))}
        self.compute_savings()
        savings_pool = self.savings.copy()
        while savings_pool:
            _, i, j = self.biased_choice(savings_pool)
            savings_pool = [s for s in savings_pool if s[1:] != (i, j)]
            ri, rj = self.client_to_route[i], self.client_to_route[j]
            if ri == rj:
                continue
            route_i, route_j = self.routes[ri], self.routes[rj]
            candidates = [
                route_i + route_j,
                route_i + list(reversed(route_j)),
                list(reversed(route_i)) + route_j,
                list(reversed(route_i)) + list(reversed(route_j)),
            ]
            merged = None
            for cand in candidates:
                if self.route_feasible(cand): # Check packing
                    merged = cand
                    break
            if merged is None:
                continue
            self.routes.append(merged)
            new_idx = len(self.routes) - 1
            for c in merged:
                self.client_to_route[c] = new_idx
            for idx in sorted([ri, rj], reverse=True):
                del self.routes[idx]
                for k in self.client_to_route:
                    if self.client_to_route[k] > idx:
                        self.client_to_route[k] -= 1
        return [[0] + [i + 1 for i in r] + [0] for r in self.routes]
```

To improve solution quality, the solver repeats the biased randomized Clarke and Wright procedure multiple times. Across iterations, the best solution is selected based on total travel distance, while also recording the number of vehicles used. This multi-start strategy allows the algorithm to exploit randomness to explore different regions of the

solution space, while maintaining the efficiency and structure of the Clarke and Wright heuristic.

Results

Due to the absence of publicly available benchmark instances specifically tailored to the Two-Level Vehicle Routing Problem with two-dimensional loading constraints, we decided to generate synthetic instances. For this purpose, an instance generator class was implemented, allowing the creation of random 2L-VRP instances based on a set of predefined parameters. These parameters include the number of clients, the maximum number of items per client, the spatial distribution of clients within a bounded grid, and the geometric and weight capacities of the vehicle. Each client is assigned a random location and a random set of rectangular items whose dimensions and weights are constrained by the vehicle specifications. This approach provides a controlled yet flexible framework for evaluating the proposed solution under different problem scales and capacity configurations.

Using this generator, multiple experiments were conducted by varying the number of clients, the number of items per client, and the vehicle dimensions. Each instance is identified using a compact naming convention of the form C*I*–W×H, where C denotes the number of clients, I the maximum number of items per client, and W×H the width and height of the vehicle loading area, respectively. This notation allows for an immediate interpretation of the instance characteristics and facilitates comparisons across different configurations.

Table 1: Computational results obtained for randomly generated 2L-VRP instances.

Instance	**Vehicles Used**	**Total Distance**	**Execution Time**
C10I2–5.0×10.0	2	395.58	0.1750 s
C20I2–8.0×6.0	5	716.12	1.2302 s
C25I3–7.5×15.0	7	796.06	1.9631 s
C50I5–15.0×10.0	25	2244.58	17.7075 s

As can be observed in Table 1, instances of varying sizes were solved within relatively short computational times. Small instances, such as C10I2–5.0×10.0, are solved almost instantaneously, while medium-sized instances remain well below a few seconds. Even for the largest tested configuration, C50I5–15.0×10.0, the total runtime remains moderate, indicating that the proposed approach scales reasonably well with respect to both routing complexity and packing feasibility checks.

Focusing on an intermediate-sized instance such as C25I3–7.5×15.0, the resulting routes are illustrated in Figure 1. The spatial distribution of the routes shows a coherent clustering of clients, with each vehicle serving geographically compact subsets, which is consistent with the expected behavior of the Clarke and Wright Savings heuristic. The number of vehicles required remains moderate, suggesting an effective trade-off between route length and capacity feasibility.

On the other hand, if we focus on the packing visualization shown in Figure 2, it can be observed that vehicle loading capacities are never fully utilized. This behavior is largely

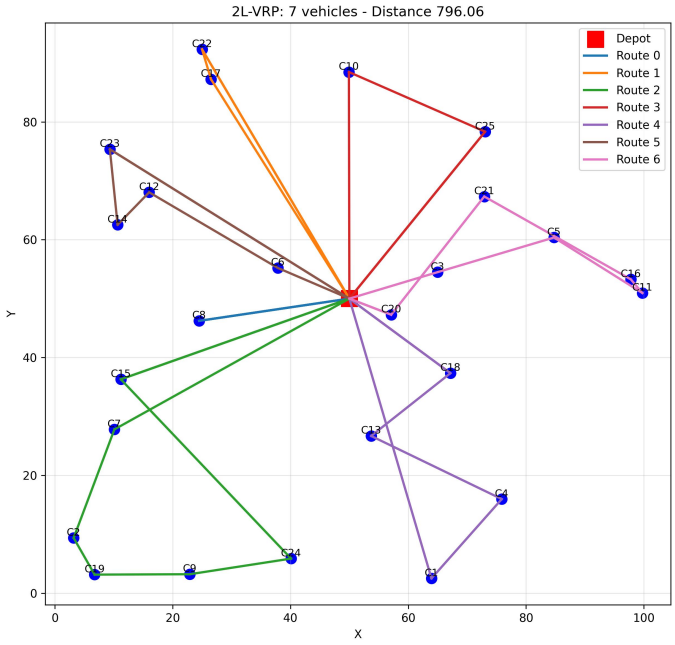


Figure 1: Routing solution for instance C25I3–7.5×15.0 obtained using the biased randomized Clarke and Wright Savings heuristic.

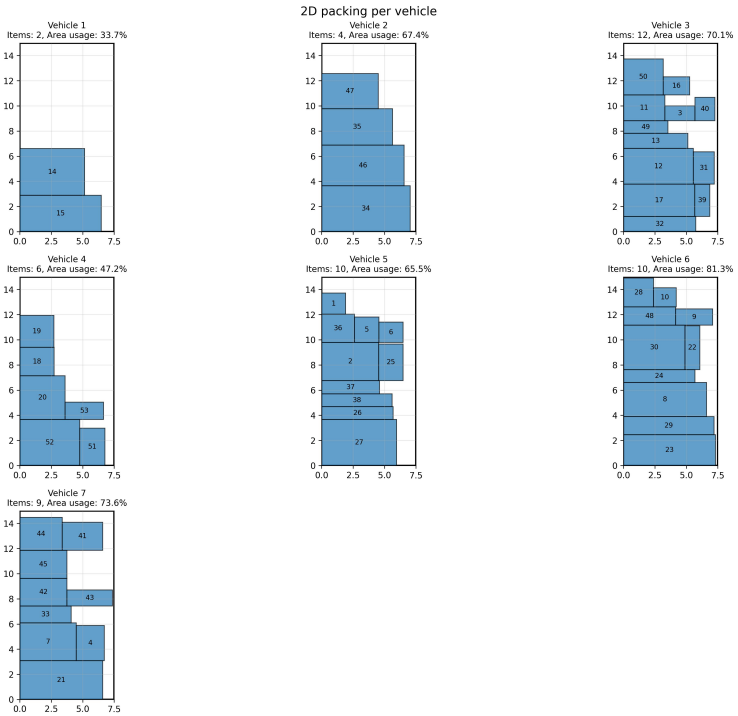


Figure 2: Two-dimensional loading configuration per vehicle for instance C25I3–7.5×15.0, illustrating item placement and unused loading space.

due to the two-dimensional loading constraints, which often become binding before the vehicle area is completely filled. In our implementation, only a basic shelf-based

heuristic is used for item placement, which limits the efficiency of space utilization. For example, in the first route, only 33.7% of the available loading area is occupied, even though additional items could theoretically fit if a more advanced packing heuristic were employed. As a result, unused space remains within the vehicles despite all assigned items being feasibly packed. This phenomenon highlights the main challenge of the 2L-VRP, where maximizing routing efficiency does not necessarily guarantee optimal loading efficiency, and underscores the importance of explicitly incorporating packing feasibility into the route construction process.

2 Unit 7. Simulation Modeling

2.1 Simulation Modeling

Simulation modeling is a methodology used to study the behavior of systems through the creation and experimentation of models that replicate aspects of real-world systems. Fundamentally, simulation involves building a simplified representation of a system — either conceptual, mathematical, or computational — and then observing its behavior under controlled experimental conditions. In this context, a model is an abstraction that represents the essential structure and dynamics of the system of interest, focusing only on the elements necessary to address predefined study objectives.

According to White and Ingalls, 2015, simulation is experimentation with a model such that the behavior of the model imitates relevant aspects of the system under study, and the user conducts experiments on the model to infer the behavior of the actual system. This framework allows researchers to explore how a system might perform without the need for direct experimentation on the real system, which can be impractical, costly, disruptive, or even unsafe. Simulation modeling is particularly valuable when analytical or closed-form solutions are unavailable or insufficient due to system complexity.

The paper illustrates the concept of simulation modeling through a call center example, which is used to demonstrate discrete-event simulation concepts. In this example, incoming telephone calls are treated as dynamic entities that flow through a call center system, interacting with activities such as queueing, waiting, and service processing. The simulation advances the state of the system by processing events (e.g., arrivals, service completions), which cause changes in system state variables (like queue lengths and waiting times). Simulating such a model enables the evaluation of different staffing or policy configurations without interrupting an actual call center's operations.

2.1.1 Monte Carlo Simulation

Monte Carlo Simulation is a class of computational simulation techniques that rely on repeated random sampling and statistical analysis to estimate the behavior of a system under uncertainty. Rather than solving a model analytically, Monte Carlo Simulation explores the range of possible outcomes by repeatedly generating random input values from specified probability distributions and observing the resulting output scenarios. This approach enables systematic what-if analysis across the full range of input variability, rather than evaluating only a few fixed scenarios such as best, worst, or base cases.

Raychaudhuri, 2008 is particularly significant because it provides a comprehensive tutorial on the methodology, practical implementation, and applications of Monte Carlo Simulation, making it accessible. The tutorial emphasizes a structured approach: starting from a deterministic "base case" model, identifying appropriate input distributions from historical data, generating random variates, and performing statistical analysis on the resulting outputs to guide decision-making. This systematic framework addresses common pitfalls of ad-hoc "what-if" analysis and highlights the advantages of a rigorous, probability-driven approach. Also, by aggregating the outputs of many such runs,

the experimenter can estimate statistical properties of the system's behavior, such as means, variances, or percentiles, and make probabilistically informed decisions based on the aggregated results.
The paper also illustrates Monte Carlo Simulation through several practical examples. One example involves reliability analysis in engineering systems, where random failure and repair times are sampled to evaluate system performance over time. Another example is drawn from business process analysis, such as using Monte Carlo methods to estimate financial risk or process variability in operational systems. These examples demonstrate how Monte Carlo Simulation provides insights into both performance and risk, enabling data-driven decisions without the need for exhaustive enumeration of all input combinations.

2.1.2 Input Modeling

Input Modelling is a core component of stochastic simulation that involves selecting, fitting, and validating probability models to represent the uncertain inputs driving a simulation model. While simulation itself focuses on the dynamics of a system under uncertainty, input modelling addresses the source and characterization of that uncertainty. The key purpose is to ensure that the random processes used within a simulation reflect real-world variability as closely as possible so that the simulation outputs provide reliable insights.
Biller and Nelson, 2002 systematically addresses the most common questions that new simulation practitioners face when constructing input models. An input model is not simply a way to inject randomness into a simulation, it is the probability distribution or stochastic process that is chosen to represent an uncertain system element, and therefore directly influences the behavior of the simulated system.
A classic example highlighted in the paper illustrates why input modelling matters. Suppose a manufacturer must estimate the lifetime of a component that fails unpredictably. If the lifetime is assumed to be a distribution with a mean of two years, but the uncertainty around that lifetime is ignored, decisions based only on the average may drastically misrepresent expected performance or cost. By explicitly modelling the lifetime as a probability distribution, the simulation can capture the variability and provide more accurate performance measures.
The paper emphasizes that choosing an input model is not merely a technical statistic exercise; it requires careful consideration of the physical context of the data, the impact of tails and rare events, and the relationships among variables. It also points out common pitfalls, such as using a distribution that matches only the mean but not the shape of the data, which may lead to biased simulation outputs even if other elements of the model are correctly specified.

2.1.3 Input Probability Distributions

Input probability distributions are fundamental building blocks of any stochastic simulation model, representing the stochastic behavior of uncertain system inputs. In discrete-event or stochastic simulation, random inputs such as interarrival times, service

times, processing durations, or demand quantities must be characterized by probability distributions so that the simulation can generate appropriate random variates during execution. Without explicit probability distributions for these inputs, a simulation cannot faithfully represent the inherent randomness of the real system, and its results may be misleading or invalid.
Biller and Nelson, 2002 highlights the critical importance of carefully selecting input probability distributions. The paper begins by showing real-world data sets, for example, machine processing times, repair times, and interarrival times, whose empirical histograms exhibit characteristics (such as skewness) that differ markedly from standard symmetric distributions like the normal distribution. These visual examples illustrate that the choice of distribution has significant implications on simulation output behavior.
The paper also identifies two major pitfalls in input distribution selection that are especially relevant:

- **Pitfall 1: Replacing a Distribution by its Mean.** Many distributions may accurately capture central tendencies like the mean and median but fail to model the extremes of the data. Ignoring the tails can underestimate rare but critical events, such as long service delays or extreme system loads, leading to overly optimistic performance estimates.

- **Pitfall Number 2: Using the Wrong Distribution.** Selecting a distribution that does not reflect the true variance of the data can distort the system's stochastic behavior. For example, assuming a normal distribution for skewed service times may reduce apparent variability, causing simulation outputs like queue lengths or waiting times to be biased.

Choosing appropriate input probability distributions is essential because incorrect assumptions about input behavior can lead to significant errors in simulation outputs. These pitfalls highlight that careful analysis, fitting, and validation are not optional steps but critical for maintaining the credibility of simulation results.

2.2 Distribution Fitting

With the aim of putting into practice the concepts of input modelling and input probability distributions discussed in the previous sections, we conducted a series of experiments to evaluate the accuracy of distribution fitting using synthetic data. The goal was to simulate a realistic scenario in which the underlying stochastic behavior of system inputs is known, and to verify whether automated fitting procedures can correctly identify both the type of distribution and its parameters. For this purpose, three standard distributions were selected: Normal, Exponential, and Log-Normal.
Synthetic data sets were generated for each distribution, and the Python library fitter was used to identify the best-fitting theoretical distribution. The sum of squared errors (SSE) was employed as the criterion for selecting the distribution that most closely approximates the generated data.

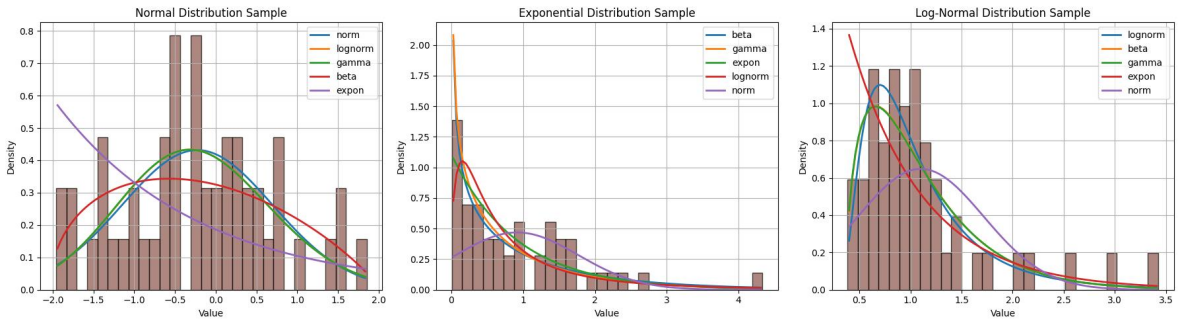


Figure 3: Fitted probability distributions for the 50-sample experiment.

In the first experiment, 50 observations were generated for each of the three distributions. For the Normal distribution, the fitting procedure successfully identified a normal curve, but the estimated mean and standard deviation were slightly off, with parameters 'loc': -0.2255, 'scale': 0.9243 compared to the true values of mean = 0 and sigma = 1. This small discrepancy is a consequence of the limited sample size, which introduces variability in the empirical data and makes parameter estimation less precise.
In contrast, the Exponential distribution was not correctly identified. The fitting algorithm suggested a Beta distribution with parameters 'a': 0.6609, 'b': 39.9051, 'loc': 0.0055, 'scale': 59.3008, resulting in a sum of squared errors of 8.7316. This misidentification illustrates how small samples can produce empirical histograms that deviate substantially from the theoretical shape, leading the fitting procedure to favor distributions that approximate the observed data rather than the true underlying distribution.
Finally, for the Log-Normal distribution, the algorithm correctly selected a log-normal curve, yet the fitted parameters ('s': 0.6475, 'loc': 0.2462, 'scale': 0.6914) differed from the true values (mean = 0, sigma = 0.5), and the sum of squared errors was 19.7807. Again, the relatively small sample size caused noticeable fluctuations in the observed data, affecting both the distribution fit and the parameter estimation.
Overall, these results illustrate a general principle in input modelling: small sample sizes can introduce significant variability in both distribution selection and parameter estimation, leading to suboptimal fits. This phenomenon is particularly evident for distributions such as Exponential, where the empirical data from 50 samples may resemble other flexible distributions like Beta.

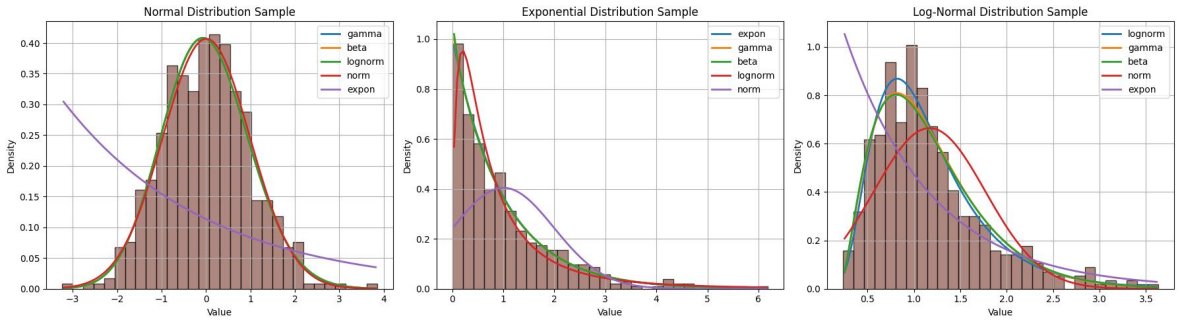


Figure 4: Fitted probability distributions for the 500-sample experiment.

To assess the effect of sample size on fitting accuracy, the experiment was repeated with larger samples of 500 observations for each distribution. With this increase in data points, the fitting procedure correctly identified the underlying distributions in all three cases, and the estimated parameters exhibited smaller deviations from the true values. This improvement occurs because larger samples reduce the influence of random fluctuations and provide a more faithful representation of the underlying probability distribution, allowing statistical fitting procedures to converge closer to the true parameters.

The impact of sample size on distribution fitting is illustrated in Figures 3 and 4. Figure 3 shows the fitted distributions for the 50-sample experiment, where deviations from the true distributions are evident, particularly for the Exponential and Log-Normal cases. Figure 4 shows the fitted distributions for the 500-sample experiment, where the fitted curves closely align with the underlying theoretical distributions, demonstrating the reduction in fitting error and improved reliability of parameter estimates.

Overall, these results reinforce the importance of adequate sample sizes when performing input probability distribution fitting for simulation models. Insufficient data can lead to misidentification or biased parameter estimates, whereas larger samples increase confidence in the selected distributions and improve the accuracy of subsequent simulation experiments.

2.3 Monte Carlo Simulation Experiments

To put into practice the concepts discussed in the previous section on Monte Carlo Simulation, we conducted a series of experiments in two different contexts. The goal of these experiments is to illustrate how Monte Carlo methods can be used to model uncertainty, estimate probabilistic outcomes, and evaluate system performance under stochastic conditions.

In both contexts, we generate random inputs according to known probability distributions, perform repeated sampling to simulate system behavior, and analyze the resulting outputs to estimate key performance measures. This approach not only reinforces the theoretical principles of Monte Carlo Simulation but also provides practical insights into its application for decision-making under uncertainty.

2.3.1 S&P Weekly Return

The first experiment aims to answer the question: "What is the likelihood that the S&P 500 index will increase by more than 5% at the end of the week?". To perform this analysis, we rely on the available historical data, which provides an average daily return of 0.03% and a daily standard deviation of 0.97%.

Using these parameters, we conducted 10,000 Monte Carlo simulations to estimate the weekly return. In each simulation, the daily return from Monday to Friday was generated as a random variate following a normal distribution, calculated using the inverse normal function `(INV.NORM(RAND(), mean, standard deviation))` with the specified daily mean and standard deviation. The weekly return was then obtained as the sum of the five simulated daily returns.

S&P Average Daily Return	0,03%	
Daily Standar Deviation	0,97%	
	Daily	Cumulative
Monday	-0,27%	-0,27%
Tuesday	-0,51%	-0,78%
Wednesday	0,45%	-0,33%
Thursday	-0,50%	-0,83%
Friday	-0,43%	-1,25%

SIMULATION

	-1,25%		
Run 1	0,05%	Average	0,12%
Run 2	-1,69%	Standard Deviation	2,16%
Run 3	-0,17%		
Run 4	-0,91%		
Run 5	1,97%	**Prob > 5%**	1,14%
Run 6	0,43%		
Run 7	1,50%		
Run 8	0,04%		
Run 9	1,57%		

Figure 5: Overview of the S&P Monte Carlo Simulation of weekly returns.

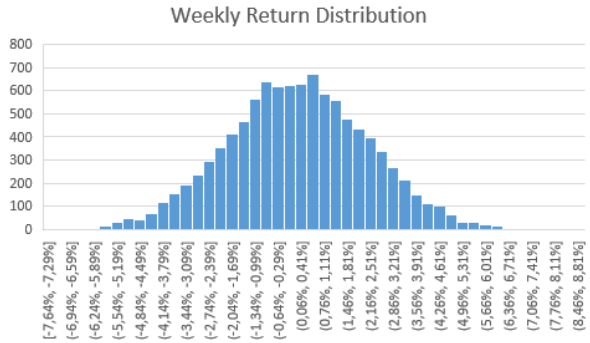


Figure 6: Histogram of simulated weekly S&P 500 returns based on 10,000 Monte Carlo simulations.

Analyzing the results, the simulated weekly returns had an average of 0.12% and a standard deviation of 2.18%, reflecting the accumulation of daily variability over the five trading days. The histogram of the simulated weekly returns, shown in Figure 6, closely follows a normal distribution, ranging approximately from -8% to +10%, as expected given the assumption of normally distributed daily returns.
Regarding the original question, the simulation results indicate that the probability of the weekly return exceeding 5% is only 1.28%. This low likelihood highlights the impact of daily volatility on weekly performance and demonstrates how Monte Carlo Simulation can be used to quantify the probability of extreme outcomes in financial markets.

2.3.2 Bookstore Order Quantity

The second experiment focuses on simulating book sales in a bookstore, taking into account both cost structure and uncertain customer demand. The model is based on the following data:

- **Cost data:** The bookstore purchases each book at a unit cost of €7.50, sells it at a unit price of €10.00, and allows a refund of €2.50 per unsold unit.

- **Demand distribution:** Customer demand is represented by a discrete cumulative probability distribution as follows: 100 books with 30% probability; 150 books with 20% probability; 200 books with 30% probability; 250 books with 15% probability; and 300 books with 5% probability

This setup captures the uncertainty in weekly demand and the corresponding impact on revenue, costs, and refunds. The unit cost, selling price, and refund values are used to compute the financial outcome for each simulated scenario, while the demand probabilities determine the likelihood of each possible sales volume.

BOOKSTORE SIMULATION MODEL

Cost data

Unit Cost	7,50 €
Unit Sell Price	10,00 €
Unit Refund	2,50 €

Demand Distribution

Cum. Prob	Demand	Probability
0%	100	30%
30%	150	20%
50%	200	30%
80%	250	15%
95%	300	5%
100%		

Decision Variable

Order Quantity	200

Summary measures for simulation

	Revenue	Cost	Refund	Profit
Average	1.600,00 €	1.500,00 €	100,00 €	200,00 €
Standard Deviation	436,11 €	- €	109,03 €	327,08 €
Minimum	1.000,00 €	1.500,00 €	- €	- 250,00 €
Maximum	2.000,00 €	1.500,00 €	250,00 €	500,00 €

Simulation

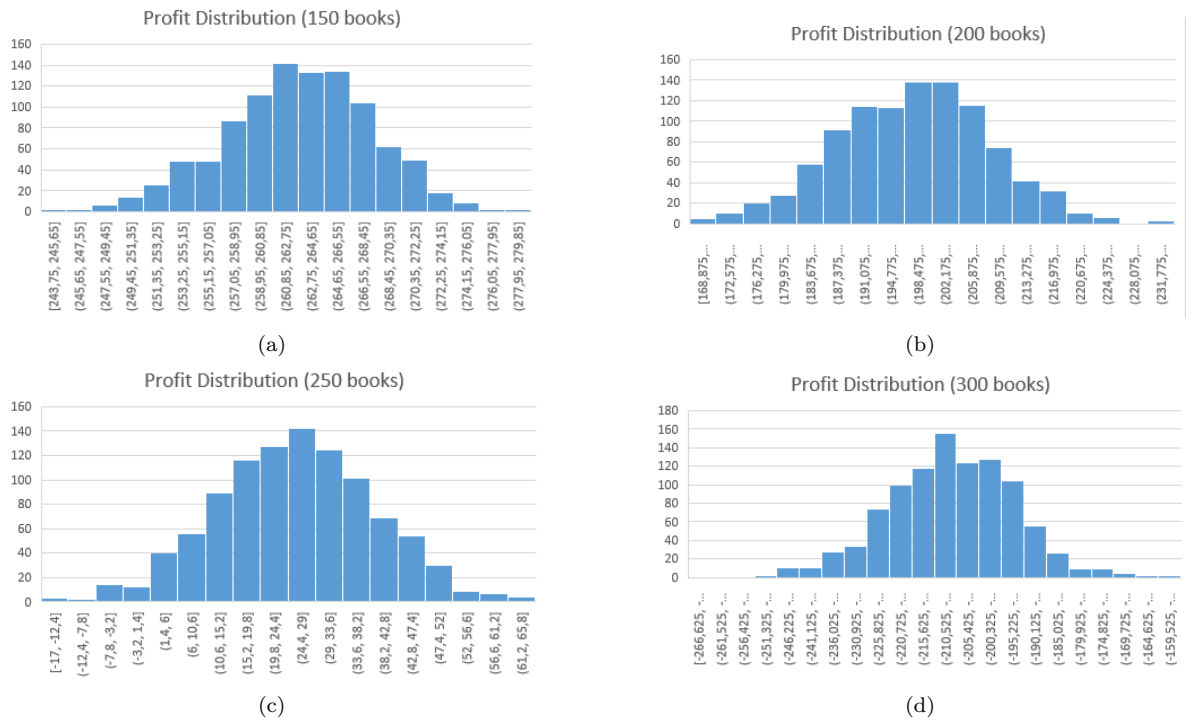
Replication	Random	Demand	Revenue	Cost	Refund	Profit
1	0,082497957	100	1.000,00 €	1.500,00 €	250,00 €	- 250,00 €
2	0,498404108	150	1.500,00 €	1.500,00 €	125,00 €	125,00 €
3	0,510441578	200	2.000,00 €	1.500,00 €	- €	500,00 €
4	0,785439185	200	2.000,00 €	1.500,00 €	- €	500,00 €
5	0,112473274	100	1.000,00 €	1.500,00 €	250,00 €	- 250,00 €
6	0,681078397	200	2.000,00 €	1.500,00 €	- €	500,00 €
7	0,94014081	250	2.000,00 €	1.500,00 €	- €	500,00 €
8	0,641487774	200	2.000,00 €	1.500,00 €	- €	500,00 €
9	0,991144733	300	2.000,00 €	1.500,00 €	- €	500,00 €
10	0,604624917	200	2.000,00 €	1.500,00 €	- €	500,00 €

Figure 7: Overview of the Bookstore Profits Monte Carlo Simulation.

To evaluate the system, we conducted 1,000 Monte Carlo simulations for each decision scenario, where the decision variable is the number of books ordered by the bookstore. Different order quantities were tested, 150, 200, 250, and 300 books, to analyze how the system behaves under varying stocking decisions. For each set of simulations, standard performance metrics were calculated, including Revenue, Cost, Refund, and Profit, allowing for a comprehensive statistical analysis of outcomes.

The results highlight interesting patterns in the system. When ordering 150 books, all simulated profits are positive, with an average of approximately €262 and a range between €240 and €280. However, as the order quantity increases, profits begin to decrease and variability grows. With 200 books, the profit range widens, while 250 books result in some negative profits, indicating overstocking relative to demand. This trend becomes even more pronounced for 300 books, where profits are consistently negative, fluctuating between -€270 and -€170.

These outcomes are illustrated in Figure 8, showing how increasing the number of books purchased can lead to lower profitability and greater financial risk. This experiment clearly demonstrates the importance of balancing inventory decisions with stochastic demand, and highlights the value of Monte Carlo Simulation in exploring the probabilistic behavior of the system under uncertainty.



(a) (b) (c) (d)

Figure 8: Histograms of simulated bookstore profits for different order quantities (150, 200, 250, and 300 books) based on 1,000 Monte Carlo simulations each.

2.4 ATM Simulation

The final experiment focuses on simulating the operations of a bank with a single service counter, aiming to evaluate customer flow, waiting times, service performance, and overall system behavior under stochastic conditions. Unlike a single-run demonstration often presented in lectures, we extended the simulation to multiple replications in order to obtain a statistical characterization of the system and capture variability in key performance indicators. The simulation was implemented in Python using SimPy.

The simulation models the following scenario: a total of 70 customers arrive at the bank according to a stochastic arrival process, with interarrival times following an exponential distribution with a mean of 10 minutes. Each customer has a random patience time uniformly distributed between 1 and 5 minutes, representing the maximum time they are willing to wait before leaving the queue. Service times are also stochastic, modeled as an exponential distribution with a mean of 12 minutes per customer. The bank has a single counter, creating a single-server queueing system, where customers who arrive while the server is busy must wait until it becomes available or abandon the queue if their patience expires.

Algorithm 1 Single Replication of Bank Simulation

Require: Total customers N, mean interarrival time λ, patience range $[P_{min}, P_{max}]$, mean service time μ

Initialize simulation environment *env*
Initialize server counter with capacity 1
for customer i in $1 \dots N$ **do**
  Generate interarrival time $t \sim \text{Exponential}(\lambda)$
  Generate patience $p \sim \text{Uniform}(P_{min}, P_{max})$
  Wait t in simulation
  if server available within p **then**
    Customer begins service
    Generate service time $s \sim \text{Exponential}(\mu)$
    Update wait time, service time, system time
  else
    Customer reneges
  end if
  Update queue length and server utilization
end for

Results

To analyze the stochastic behavior of the system, 1,000 independent replications of the simulation were performed, each with the same parameters but different random seeds. This approach allows the computation of mean, standard deviation, minimum, maximum, and 95% confidence intervals for all key metrics, providing a comprehensive statistical understanding of system performance rather than relying on a single realization.

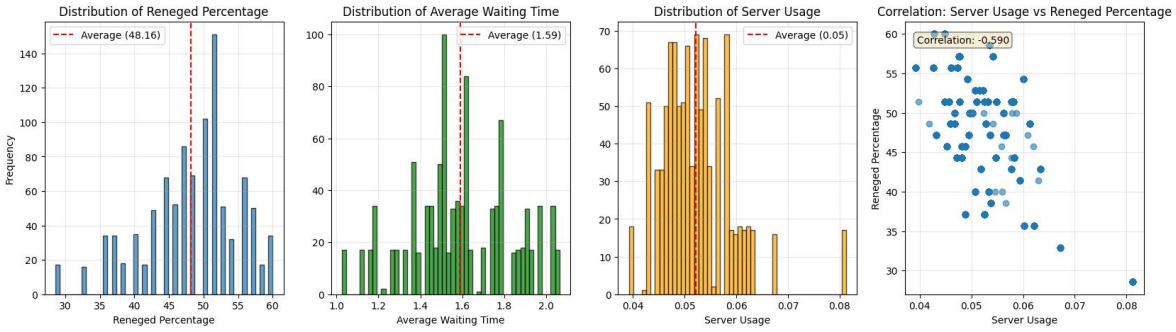


Figure 9: Distributions and correlation of performance metrics in the bank simulation based on 1,000 Monte Carlo replications.

The results of the bank simulation provided in the Figure 9, provide a detailed understanding of the system's behavior under stochastic conditions. First, the percentage of customers who reneged ranges from approximately 30% to 60%, with an average of 48.16%. This indicates that nearly half of the customers were unable to wait for service

and abandoned the queue, highlighting the impact of limited patience relative to the arrival and service rates.

On the other hand, the average waiting time for all customers is only 1.6 minutes (about 1 minute and 36 seconds). This relatively short waiting time suggests that, although a significant fraction of customers leave the queue, those who are served experience minimal delay, reflecting the balance between service rate and customer arrivals in this single-server system.

Focusing on server utilization, the average usage is around 0.05, which indicates that the teller is idle most of the time. This low utilization may appear counter-intuitive given the high reneging percentage, but it occurs because customers with low patience often leave the queue before being served, leaving the server frequently unoccupied.

Finally, the correlation between server utilization and reneged percentage is negative, with a value of -0.59, suggesting that higher utilization is associated with fewer customers abandoning the system. In other words, when the server is more consistently busy, more customers are able to receive service before their patience expires, demonstrating a clear trade-off between service capacity and customer abandonment in queueing systems.

3 Unit 8. Simheuristics

Simheuristics are a class of metaheuristic methods that integrate simulation techniques into heuristic or metaheuristic optimization frameworks. The main idea behind simheuristics is to address optimization problems under uncertainty, where some problem parameters—such as demand, processing times, travel times, or resource availability—are not deterministic but stochastic.
Traditional heuristics or metaheuristics, such as Tabu Search or the Clarke & Wright Savings algorithm, typically assume that all input data is deterministic. However, in many real-world problems, ignoring uncertainty can lead to solutions that are infeasible or suboptimal when randomness is present. Simheuristics address this gap by combining the exploratory power of heuristics with the evaluative strength of simulation.
The general workflow of a simheuristic algorithm involves three main steps:

1. **Solution Generation:** The heuristic or metaheuristic explores the solution space, generating candidate solutions based on the problem structure.

2. **Solution Evaluation via Simulation:** Instead of evaluating candidates deterministically, the algorithm uses simulation to estimate the performance of each solution under stochastic conditions. This allows the method to capture expected costs, variability, and risk of constraint violations.

3. **Iterative Improvement:** The heuristic uses the simulation feedback to guide the search, favoring solutions with better expected performance under uncertainty.

By combining these steps, simheuristics can efficiently handle stochastic combinatorial optimization problems, such as vehicle routing under uncertain travel times, inventory management with stochastic demand, or scheduling with random processing times.
For example, in the context of a vehicle routing problem with stochastic customer demands, a simheuristic can generate candidate routes using a traditional savings algorithm and then simulate the loading and delivery process multiple times to evaluate each route's feasibility and expected cost under demand variability. The algorithm then iteratively refines the solutions based on this feedback, producing robust routes that perform well under uncertainty.

3.1 Simulation-based optimization to enhance logistics

Raba et al., 2019 investigate how the integration of Internet of Things (IoT) technologies with simulation-based optimization can enhance logistics performance in agri-food supply chains, with a particular focus on animal feed distribution. Feed supply chains are characterized by stochastic demand, limited storage capacity, and high transportation costs, which makes efficient replenishment planning crucial. Traditional optimization models often ignore uncertainty, whereas pure simulation approaches lack decision-making power. To address these limitations, the authors propose a hybrid simulation–optimization framework that leverages real-time inventory data collected via IoT sensors.

The problem is modeled as a multi-period inventory routing problem (IRP) with stochastic demands, where both replenishment quantities and vehicle routes must be jointly optimized over a finite planning horizon. Each farm silo is equipped with smart sensors that provide continuous measurements of inventory levels. These data are incorporated into a simheuristic approach combining biased-randomized metaheuristics with Monte Carlo simulation. The optimization component generates candidate refill policies and routing plans, while the simulation component evaluates their expected performance under demand uncertainty. This framework allows refill strategies to be dynamically updated as new information becomes available, creating a reactive decision-support system.
The proposed method is tested through extensive computational experiments using 27 benchmark instances with up to 80 demand nodes, multiple planning horizons (3, 5, 7, and 10 periods), and varying demand levels and variances. Numerical results show that the simheuristic approach substantially reduces total expected costs compared to baseline homogeneous replenishment policies. On average, cost reductions range from approximately 30% to over 50%, depending on demand intensity and planning horizon. For low-demand scenarios (average demand around 5–15% of silo capacity), reductions exceed 40%, while in extreme low-demand cases (around 5%), savings reach over 50% across all horizons. As expected, total costs increase with higher demand variance, but the proposed approach consistently outperforms baseline strategies.
Score: 9

3.1.1 Implementation

The implemented Simheuristic framework addresses a stochastic inventory routing problem (IRP) by combining heuristic solution methods with simulation-based evaluation to handle uncertainty in customer demand. The process begins by initializing the problem instance with a depot and a set of customers. The demand for each customer in each period is modeled as a log-normal distribution, allowing the system to capture stochastic fluctuations while respecting storage constraints. Each customer is also assigned a maximum storage capacity and an initial stock level, which guide the replenishment decisions.
The algorithm starts by generating an initial solution, in which each customer is assigned replenishment quantities equal to their expected demand. This deterministic plan is then evaluated using simulation across multiple stochastic demand realizations, producing estimates for both inventory and routing costs.
Once the initial solution is obtained, the algorithm enters a multi-start search procedure. Candidate solutions are generated through biased randomization, which perturbs the replenishment quantities around the values of existing solutions while respecting storage limits. These candidates are evaluated using simulation, and the best-performing solutions are retained in an elite set. Over successive iterations, new solutions are generated from this elite set, gradually improving both the total cost and robustness of the plan.

Listing 2: Python implementation of the Simheuristic for the IRP.

```
class SimheuristicIRP:
```

```
...
def multi_start_search(self, initial_solution: IRPSolution,
                       num_starts: int = 50) -> tuple[IRPSolution, list[IRPSolution]]:
    best_solution = initial_solution
    elite_solutions = [initial_solution.copy()]
    for iteration in range(num_starts):
        if iteration < len(elite_solutions):
            base = elite_solutions[iteration % len(elite_solutions)]
        else:
            base = random.choice(elite_solutions)
        new_solution = self.biased_randomization(base)
        new_solution.total_cost = self.evaluate_solution(new_solution, use_simulation=True)
        self.cost_history.append(new_solution.total_cost)
        elite_solutions.append(new_solution)
        elite_solutions.sort(key=lambda s: s.total_cost)
        elite_solutions = elite_solutions[:self.num_elite_solutions]
        if new_solution.total_cost < best_solution.total_cost:
            best_solution = new_solution
        self.best_cost_history.append(best_solution.total_cost)
        if (iteration + 1) % 10 == 0:
            print(f"Iteration {iteration + 1}: Best cost = {best_solution.total_cost:.2f}")
    self.elite_costs_history = [sol.total_cost for sol in elite_solutions]

    return best_solution, elite_solutions
```

For routing, each set of replenishment quantities is processed by a heuristic vehicle routing procedure inspired by the Clarke & Wright savings algorithm. Routes are initially assigned individually to each customer and then iteratively merged based on savings and vehicle capacity constraints, ensuring that no vehicle is overloaded. The total distance traveled by the fleet is used as the routing cost, which is combined with inventory costs from the simulation to calculate the overall cost of the solution.
Finally, the refinement stage reevaluates the elite solutions using a higher number of simulation runs to obtain statistically robust estimates of total cost, ensuring that the final solution is not biased by stochastic variations in demand. The algorithm outputs the final replenishment plan, routes for each period, and a cost breakdown, as well as visualizations showing route structures and stochastic demand distributions for selected customers.

Results

For the experiments, predefined VRP instances are loaded into the system, including the depot and customer locations. Then, the demand for each customer in each period is generated according to a log-normal distribution, scaled by a factor γ and a variance level. This stochastic modeling allows the algorithm to estimate expected inventory shortages and holding costs, making it suitable for real-world applications where exact customer demand is unknown.
Focusing on a specific instance generated using the routes of "A-n32-k5" with holding_cost = 0.5, a planning horizon of 5 periods, gamma = 1, a variance level of 2, and 5 vehicles, the final solution provides a clear picture of how the system balances inventory and routing under stochastic demand.
For the previous instance the system generates an initial cost of 19,615.85, which is reduced to a final cost of 18,015.55, yielding a total saving of 1,600.30 or 8.2%. The total replenishment quantity across all customers is 2,050 units, with an average of 13.23 units per customer, reflecting a balanced allocation that meets demand while avoiding excessive inventory. The inventory cost contributes the majority of the total cost at 13,447.80, capturing both holding costs and penalties for potential stockouts, while the routing cost is 4,567.75, accounting for the distances traveled by the fleet to service all customers.

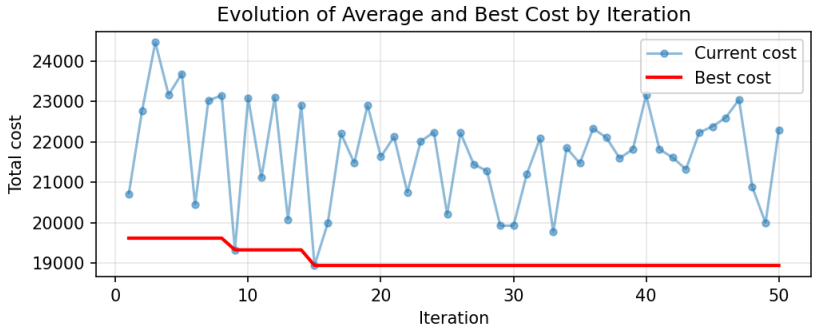


Figure 10: Evolution of the total cost over iterations during the multi-start simheuristic process for the IRP.

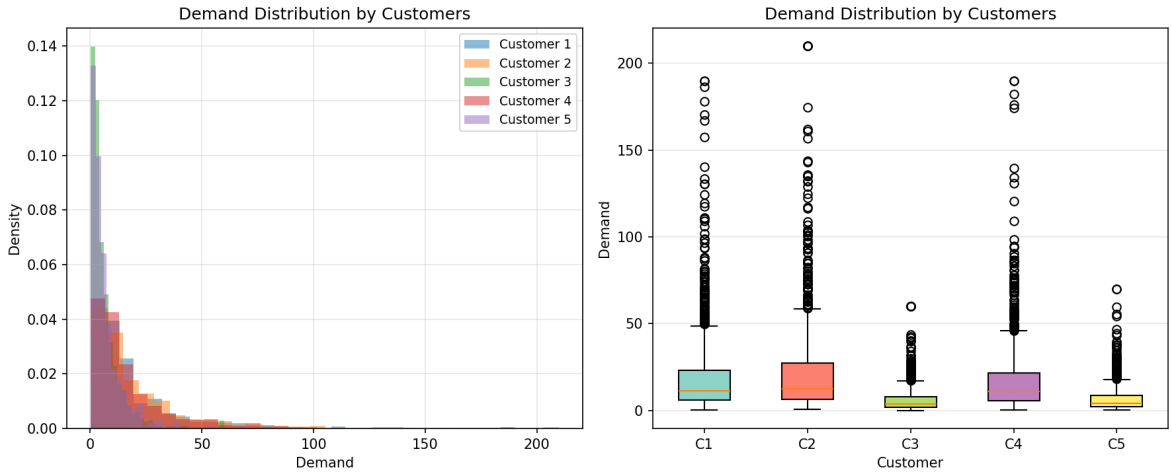


Figure 11: Demand distribution for the first five customers of the IRP.

First, we analyze the convergence process of the algorithm. The Figure 10 shows that the best solutions are found very early, typically around iteration 10, indicating that the multi-start search is effective at quickly identifying high-quality solutions. When observing the average cost per iteration, no clear pattern of improvement is visible, which is expected due to the stochastic nature of the system: each evaluation involves random demand realizations that can cause fluctuations in the cost. However, a clear reduction in variability can be seen, as the cost tends to stabilize around lower values as the algorithm progresses, reflecting increased reliability and robustness of the solutions. Turning to the solution itself, the demand distribution for the first five customers (Figure 11) illustrates the stochastic component of the problem. As expected, all customers follow a log-normal distribution, with boxplots showing variation in the realized demand for each customer. While the median values are similar, the range of possible demands differs slightly between customers, highlighting the impact of uncertainty on inventory planning and the importance of considering multiple scenarios during solution evaluation.

Finally, examining the routes for period 1 (Figure 12), we observe a geographically coherent distribution of routes. Each vehicle is assigned a subset of customers that

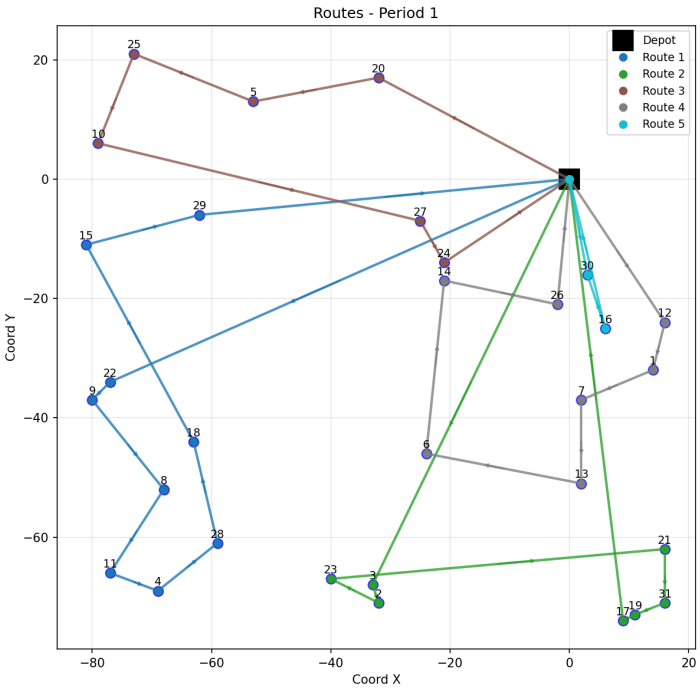


Figure 12: Geographical representation of vehicle routes for period 1.

are spatially close to each other, minimizing travel distance while respecting vehicle capacity constraints. The depot serves as the central hub, and all routes start and end there. The visualization confirms that the heuristic effectively merges delivery points to reduce total routing cost while maintaining feasible loads, and the arrow markers indicate the sequence of visits within each route. This route assignment demonstrates a balanced trade-off between efficient delivery and adherence to inventory constraints, showing how the simheuristic handles both routing and stochastic replenishment in an integrated manner.

3.2 Inventory-routing problem with stochastic demand and stock-out

Onggo et al., 2019 address the Inventory Routing Problem (IRP) under stochastic demand and potential stock-outs, highlighting the risks of relying on deterministic assumptions in complex supply chain decision-making. As inventory management and transportation planning are increasingly integrated, ignoring demand uncertainty may lead to underestimated costs and suboptimal operational plans. The authors focus on a single-period IRP motivated by vendor-managed inventory systems, where a central warehouse supplies multiple distribution centers (DCs) facing random customer demand.

Methodologically, the problem is formulated as a stochastic IRP in which replenishment levels and vehicle routes are jointly optimized to minimize the expected total cost, including routing costs, inventory holding costs, and stock-out penalties. Stock-outs are explicitly modeled through a penalty equivalent to the cost of an emergency

round-trip shipment. To solve this NP-hard problem, the authors propose a three-stage simheuristic framework that integrates Monte Carlo simulation with heuristic optimization. First, simulation is used to estimate expected inventory costs for a discrete set of replenishment policies (0%, 25%, 50%, 75%, and 100% of capacity) at each DC. Second, a multi-start biased-randomized heuristic explores combinations of replenishment policies and routing plans. Finally, an intensification stage refines the best solutions and performs a risk analysis by estimating cost variability.

The approach is evaluated on 27 benchmark instances with 32 to 80 nodes, assuming lognormally distributed demands with four variance levels (from deterministic to high variance). Numerical results show that the proposed simheuristic consistently outperforms the best-known solutions from the literature across all variance scenarios. Average total cost improvements range from approximately 1% to 6%, depending on the instance and demand variance. More importantly, the experiments demonstrate the substantial risk of ignoring uncertainty: when demand variance is 0.25, deterministic models underestimate inventory costs by more than 100% on average, while for high variance (0.75), the real inventory cost can exceed deterministic estimates by a factor of four due to frequent stock-outs.

Score: 8

4 Unit 9. Learnheuristics and Agile Optimization

4.1 Learnheuristics

Learnheuristics refers to a class of advanced optimization approaches in which statistical learning and machine learning techniques are explicitly integrated into metaheuristic algorithms with the objective of improving their performance, adaptability, and decision-making capability. The central idea behind learnheuristics is to allow the optimization process to learn from experience.
Learnheuristics are particularly well suited for industrial and business optimization problems characterized by high complexity, uncertainty, and large-scale decision spaces. Typical application domains include logistics networks, supply chain coordination, transportation planning, production systems, energy management, and financial decision support. In these settings, optimization algorithms must repeatedly solve similar problem instances under changing conditions, generating substantial amounts of historical data during their execution.
An example of use is in logistics and particularly in the Vehicle Routing Problem (VRP), where a machine learning component can be directly embedded within the optimization process to enhance decision making. In a learnheuristic VRP framework, the metaheuristic remains responsible for exploring the solution space, while a machine learning model acts as a learning layer that exploits information generated during previous iterations or past problem instances.
Concretely, as the VRP algorithm constructs and improves routes, it produces large amounts of data describing partial solutions, complete routes, move evaluations, and their associated costs. A supervised or reinforcement learning model can be trained on this data to estimate the quality of routing decisions, such as the expected cost impact of inserting a customer into a given route position, merging two routes, or applying a specific local search move. These predictions are then used to bias the search toward more promising decisions, for example by ranking candidate moves, filtering out low-potential neighborhoods, or dynamically selecting the most effective operators.

Learnheuristic for the Multi-Depot Vehicle Routing Problem

Calvet et al., 2016 explore how statistical learning techniques can be combined with metaheuristic optimization to improve decision-making in complex stochastic optimization problems, with a particular focus on logistics and routing applications. Traditional metaheuristics are powerful for solving large-scale combinatorial problems but often rely on repeated costly simulations to evaluate solutions under uncertainty. This results in high computational effort and limited scalability. The authors address this limitation by proposing a hybrid framework that integrates statistical learning models to approximate simulation outcomes, thereby reducing computational burden while preserving solution quality.
Methodologically, the paper introduces a learning-enhanced simheuristic approach. The key idea is to replace part of the simulation-based evaluation process with surrogate models built using statistical learning techniques, specifically regression-based predic-

tors. During the search process, a set of candidate solutions generated by a metaheuristic is evaluated through Monte Carlo simulation. The resulting data are then used to train predictive models that estimate expected costs and risk measures for new solutions. These learned models guide the metaheuristic by filtering and ranking candidate solutions before simulation is applied, allowing computational effort to be concentrated on the most promising regions of the solution space. The framework is tested on stochastic vehicle routing and inventory-routing-type problems with uncertain demands.

Computational experiments are conducted on benchmark instances with up to 100 customers and stochastic demand modeled through multiple probability distributions. Numerical results show that the hybrid learning-based approach achieves performance levels comparable to full simulation-based methods while substantially reducing computation time. In particular, the number of simulation runs required during the optimization process is reduced by approximately 50–70%, depending on the instance size. Despite this reduction, the final solutions differ by less than 1–3% in expected total cost compared to solutions obtained using pure simheuristics. Moreover, the learning-guided approach improves convergence speed, identifying high-quality solutions earlier in the search process.

In conclusion, the paper demonstrates that combining statistical learning with metaheuristics is an effective strategy for tackling stochastic optimization problems. The proposed approach significantly lowers computational requirements without sacrificing solution quality, making simheuristic methods more scalable and practical for real-world logistics applications. The study also highlights the potential of data-driven optimization frameworks and suggests future research directions involving more advanced machine learning models and adaptive learning mechanisms.

Score: 9

4.2 Discrete Event Heuristics

Discrete Event Heuristics (DEH) refer to a class of optimization approaches that explicitly combine discrete-event simulation (DES) with heuristic algorithms in order to solve complex decision-making problems in dynamic and stochastic systems. The core idea behind DEH is to evaluate and guide heuristic decisions using a simulation model that represents the evolution of the system as a sequence of discrete events (such as arrivals, departures, machine failures, service completions, or inventory replenishments). Rather than optimizing a static and deterministic representation of the problem, DEH operate on a more realistic model in which system states change over time in response to events.

Discrete Event Heuristics are particularly suitable for industrial and business environments where system performance depends on timing, resource interactions, and uncertainty. Typical application domains include service systems, manufacturing and production planning, healthcare operations, transportation systems, supply chains, call centers, and financial operations. In these contexts, analytical models are often insufficient to capture congestion effects, queue dynamics, breakdowns, or customer behavior, making discrete-event simulation a natural evaluation tool for candidate solutions gen-

erated by heuristics.
A representative application of DEH can be found in service and logistics systems characterized by queues and time-dependent congestion, such as distribution centers or urban transportation hubs. In these settings, a heuristic algorithm generates alternative operational policies—such as vehicle dispatching rules, resource allocation strategies, or scheduling decisions—while a discrete-event simulation model is used to evaluate their performance in terms of waiting times, service levels, resource utilization, and costs. The simulation feedback is then used to guide the heuristic search, allowing it to favor decisions that perform well under realistic, time-evolving conditions.

DER for dynamic home service routing

Fikar et al., 2016 explore how a discrete-event driven metaheuristic can be utilized to solve dynamic routing and scheduling problems in the home service industry, specifically focusing on scenarios that integrate trip sharing and walking. Traditional home service operations often assign a separate vehicle to each staff member, leading to low vehicle utilization and high transport costs. While trip sharing can reduce fleet sizes, its implementation is hindered by the complexity of synchronizing arrival times of vehicles and staff at various locations, especially in dynamic environments where cancellations or new requests occur in real-time.
The paper introduces a discrete-event driven metaheuristic (DER) that iteratively generates solutions based on the chronological occurrence of events. The algorithm manages an event list comprising vehicle pickups, deliveries, and agent scheduling. To explore the solution space effectively, the framework employs biased-randomization techniques, which introduce controlled randomness into the constructive procedure to prioritize more promising candidates. Key features of the approach include a savings-based clustering procedure to identify efficient walking routes between close customers and a decision model that evaluates whether idle vehicles should wait en-route or return to the depot to minimize unproductive time. This event-based structure avoids time-consuming "back-rolling" issues by making all timing decisions in chronological order.
The experiments were conducted on 30 real-world benchmark instances based on home health care operations in urban and sub-urban areas, with job counts ranging from 75 to 125. The DER algorithm was compared against a sophisticated static matheuristic (TS-SPBS). Numerical results demonstrate that the DER approach is exceptionally fast, finding feasible solutions in an average of 0.010 seconds. Notably, the TS-SPBS algorithm failed to find any feasible solution in 12 out of 30 instances (40%) within a one-minute runtime limit. While solution quality remains competitive—with an average gap of only 1.71% for the best-cost solution and 0.74% for average-cost solutions compared to the matheuristic, the DER algorithm significantly outperformed TS-SPBS in sub-urban instances where synchronization is more complex.
Score: 7

4.3 Agile Optimization

Agile Optimization refers to a class of optimization approaches designed to operate effectively in highly dynamic, uncertain, and rapidly changing environments, where problem parameters, constraints, and objectives may evolve over time. The fundamental principle of Agile Optimization is flexibility: instead of seeking a single optimal solution for a fixed problem formulation, agile methods focus on continuously adapting solutions as new information becomes available or as the system state changes.

Agile Optimization is particularly relevant in industrial and business contexts where decisions must be updated frequently and where responsiveness is more valuable than long computation times or theoretically optimal solutions. Typical application domains include supply chain management, logistics and transportation systems, production planning, workforce scheduling, energy systems, and real-time decision support. In these environments, disruptions such as demand fluctuations, delays, machine failures, or market changes require optimization algorithms that can quickly react and re-optimize without restarting the process from scratch.

An illustrative context for Agile Optimization is logistics and transportation planning, where routing, scheduling, and resource allocation decisions must be continuously revised in response to real-time events. For example, in a vehicle routing setting, new customer requests, traffic conditions, or vehicle breakdowns may arise after routes have already been planned. An agile optimization framework addresses this by maintaining a pool of feasible solutions and incrementally adjusting them as changes occur, rather than recomputing routes entirely.

Real-time facility location problem in Internet of Vehicles networks

Martins et al., 2022 tackle the need for real-time optimization methods in highly dynamic networked systems, specifically focusing on a variant of the Uncapacitated Facility Location Problem (UFLP) in the context of the Internet of Vehicles (IoV). The traditional UFLP is an NP-hard combinatorial optimization problem widely used in logistics and supply networks for strategic facility placement, where decisions are static and long-term. However, emerging application domains such as IoV—where vehicle demands and network conditions change rapidly over time—require frequent reoptimization to maintain quality of service (QoS).

To address this dynamic facility location problem, the authors propose an agile optimization algorithm that extends traditional biased-randomized heuristics with parallel execution and rapid reoptimization capabilities. This methodology centers on an agile optimization (AO) framework in which multiple instances of a biased-randomized heuristic run concurrently, each with different parameterizations, to quickly generate and evaluate a diverse set of RSU assignment solutions under real-time IoV conditions. The algorithm was tested using existing benchmark instances derived from classical facility location datasets adapted to IoV characteristics. Experimental results indicate that the agile optimization approach quickly produces high-quality solutions even for large problem instances, making it suitable for dynamic IoV scenarios where mapping vehicles to RSUs must be updated frequently. Compared to traditional greedy heuristics, which typically yield solution cost gaps between about 2% and 8% relative to best

known solutions (BKS), the agile optimization algorithm achieved gaps lower than 2% in most cases, thereby combining computational speed with near-optimal cost performance. In conclusion, this study demonstrates that agile optimization methods are effective for real-time facility location problems in IoV environments, delivering fast, high-quality solutions suitable for systems with stringent latency and dynamic demand patterns.
Score: 9

5 Unit 10. Genetic Algorithms

Genetic Algorithms (GAs) are a family of search and optimization algorithms inspired by the principles of Darwinian natural evolution. They are particularly useful for solving complex optimization problems where traditional methods struggle.
At their core, GAs maintain a population of candidate solutions, often called individuals or chromosomes, which evolve over successive generations. Each individual encodes a potential solution to the problem, typically represented as a sequence of values (the genotype) that can be mapped to the actual solution (phenotype) for evaluation.
The main steps of a GA are:

1. **Initialization:** A population of individuals is generated, often randomly, to represent a diverse set of potential solutions.

2. **Evaluation (Fitness Function):** Each individual is evaluated according to a fitness function, which quantifies the quality of the solution with respect to the optimization objective. Higher fitness values indicate better solutions.

3. **Selection:** Individuals are selected to become parents of the next generation. The selection process favors better solutions but may allow less-fit individuals to maintain diversity. Common selection mechanisms include:

   - **Roulette Wheel Selection:** Individuals are chosen probabilistically based on their relative fitness.
   - **Universal Sampling:** A deterministic variant of roulette selection that ensures more even representation of individuals.
   - **Tournament Selection:** Groups of individuals are randomly sampled, and the best in each group is chosen as a parent.
   - **Fitness Scaling:** Raw fitness values are transformed using a linear or nonlinear function (e.g., scaled fitness $= a \cdot$ raw fitness $+ b$) to adjust selection pressure.

4. **Crossover (Recombination):** Selected parents produce offspring by mixing their genes, combining the information from two or more solutions to create new candidate solutions. Several crossover strategies exist:

   - **Single-point or Multi-point Crossover:** One or more points in the chromosome are chosen, and segments are exchanged between parents.
   - **Uniform Crossover:** Each gene is independently selected from one of the parents.
   - **Ordered Crossover:** Used for permutation-based problems (e.g., routing), where the relative order of elements must be preserved.

5. **Mutation:** Random changes are applied to one or more genes of an offspring to maintain diversity in the population and prevent premature convergence. This introduces novel solutions that may improve performance in subsequent generations.

6. **Elitism:** To ensure that the best solutions are not lost, a subset of top-performing individuals is directly carried over to the next generation. This mechanism accelerates convergence and stabilizes the search.

7. **Iteration:** The new generation replaces the old population, and the process repeats until a stopping criterion is met, such as reaching a maximum number of generations or achieving an acceptable fitness level.

Through these mechanisms, the population of solutions gradually evolves toward higher fitness. Crossover explores the solution space by combining good traits from different individuals, mutation maintains diversity, and selection ensures that superior solutions guide the evolution. By operating on a population rather than a single solution, GAs are able to approximate global optima even in highly complex or multimodal search spaces.

5.1 One-Max Problem

The One-Max problem is a classical benchmark in the field of genetic algorithms. It is a simple yet effective problem designed to evaluate the performance of optimization algorithms. The problem consists of a binary string of fixed length where each position in the string can take the value 0 or 1. The objective of the problem is straightforward: maximize the total number of ones in the string. Mathematically, the fitness function is simply the sum of all bits in the string, meaning that the optimal solution is the string consisting entirely of ones. Despite its simplicity, the One-Max problem is widely used as a testbed to understand convergence behavior, the effect of genetic operators, and the exploration-exploitation balance of genetic algorithms.

Implementation

To solve the One-Max problem, a Genetic Algorithm (GA) was implemented. The GA is an iterative metaheuristic inspired by Darwinian natural selection and evolution. It maintains a population of candidate solutions (chromosomes), evaluates their fitness, and generates new generations via evolutionary operators. The methodology used can be described as follows:

1. **Initialization:** A population of 100 individuals was generated randomly, each represented as a binary string of length 20. The fitness of each individual was calculated using the sum of ones in the string.

2. **Selection:** Individuals for reproduction were selected using a fitness-proportionate selection approach (also called roulette-wheel selection). In this method, the probability of being chosen as a parent is proportional to the individual's fitness. This ensures that higher-performing solutions have a greater chance of passing their genes to the next generation.

3. **Crossover:** Selected parents were paired, and a single-point crossover was applied. A crossover point was randomly chosen along the string, and segments

from both parents were exchanged to produce two offspring. This operator allows the algorithm to combine promising traits from different solutions, promoting exploration of the solution space.

4. **Mutation:** Each offspring underwent bit-flip mutation with a very low probability (0.001 per bit). Mutation introduces random variation into the population, helping to maintain diversity and preventing premature convergence to local optima.

5. **Replacement and Iteration:** The offspring population replaced the previous generation, and fitness values were recalculated. The algorithm then tracked both the best solution and the average fitness in the population across 100 generations. This iterative process allowed the population to gradually evolve toward the optimal solution.

Results

The experimental setup consisted of a One-Max problem with a binary string of length 20 that was random initialized.

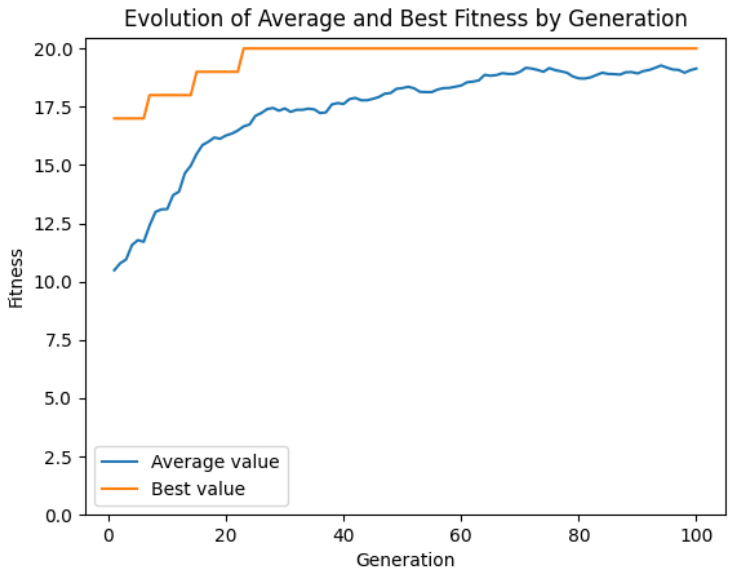


Figure 13: Evolution of average and best fitness values for the One-Max problem.

The GA successfully solved the One-Max problem. As can be seen in Figure 13 early in the process the best solution was achieved. We can see that the best fitness progressively increased from 17 ones, then 18, 19, and finally reached the optimal solution of 20 ones, corresponding to the string [1, 1, 1, ..., 1].
The convergence plot provides additional insights. While the average fitness increased gradually over generations, reflecting the improvement of the overall population, the best fitness achieved jumps at specific generations as new optimal individuals were

discovered. This behavior is consistent with the stochastic nature of the GA, crossover and mutation generate new high-quality solutions, but the average performance of the population improves more slowly due to variability.

5.2 Knapsack Problem

The Knapsack Problem is a classic combinatorial optimization challenge that introduces complexity through resource constraints. The Knapsack Problem involves a set of items, each characterized by a specific weight and value. The goal is to select a subset of these items to maximize the total value while ensuring the cumulative weight does not exceed a predefined maximum capacity.

Implementation

In this implementation, each candidate solution, or chromosome, is represented as a binary string of length equal to the number of items. A "1" indicates that the corresponding item is included in the knapsack, whereas a "0" indicates exclusion. The fitness function evaluates each chromosome by summing the values of the selected items, while assigning a fitness of zero to solutions that violate the capacity constraint. This ensures that the algorithm focuses on feasible solutions.
The GA begins by generating an initial population of random solutions. At each generation, the algorithm follows a series of steps designed to evolve the population toward better solutions:

1. **Selection:** Candidate parents for the next generation are chosen based on their fitness. Two selection mechanisms are implemented: roulette wheel selection, where the probability of being chosen is proportional to fitness, and tournament selection, where a small subset of solutions competes and the best among them is selected. Tournament selection is particularly effective in maintaining diversity while favoring higher-quality solutions.

2. **Crossover:** Selected parents are paired to produce offspring through recombination. Both one-point and two-point crossover strategies are implemented. In one-point crossover, a single cut point is selected, and segments of the parent chromosomes are swapped to create two children. In two-point crossover, two cut points define a segment to be exchanged. This operator allows the algorithm to combine good traits from different parents and explore new regions of the solution space.

3. **Mutation:** To maintain diversity and avoid premature convergence, a small probability of mutation is applied to each gene in the offspring chromosomes. Mutation flips a bit from 0 to 1 or from 1 to 0, introducing random variations that can help escape local optima and explore alternative high-value solutions.

4. **Elitism:** To ensure that the best solutions found so far are not lost, the top-performing chromosomes in each generation are preserved and carried forward

directly to the next generation. In this implementation, the top 5% of the population are considered elite.

After generating offspring through crossover and mutation, the new population is evaluated to check if the capacity constraints are not broke, and then the process is repeated for a fixed number of generations.

Results

For the GA experiment on the Knapsack Problem, the instance was generated randomly with 20 items, each assigned a weight and a value within the range of 1–200. The knapsack capacity was set to approximately 25% of the total weight of all items, resulting in a moderately constrained problem that allows for multiple feasible combinations while maintaining significant combinatorial complexity. This setup provides a challenging yet solvable instance for the genetic algorithm.
The GA reached a best global value of 1172 with a total weight of 445, as shown in the specific solution vector [0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0]. This means that a total of 7 of the 20 items were included in the knapsack. These results indicate that the GA successfully identified a combination of items that maximizes the total value while respecting the capacity constraint of the knapsack.

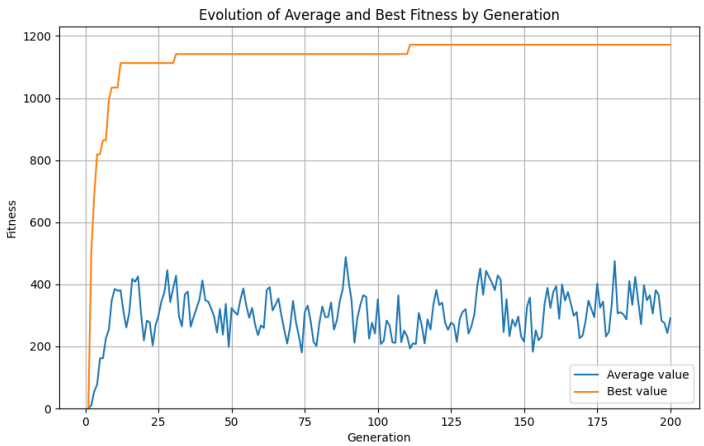


Figure 14: Evolution of average and best fitness values for the Knapsack Problem with 20 random items with weights and values within the range 1-200.

Beyond the numeric results, the evolution of the population can be analyzed through the Figure 14. The best value demonstrates a stepwise improvement, it remains constant for several generations and then increases sharply in discrete jumps. These jumps correspond to the introduction of superior solutions through crossover and mutation, which replace less fit individuals and establish new elite solutions. This behavior illustrates the exploratory nature of the GA, where significant improvements often occur suddenly rather than gradually.
In contrast, the average fitness of the population does not show a consistent upward trend. Initially, the average value increases from zero to a value in the range of approx-

imately 200–400 as low-quality solutions are gradually replaced by fitter individuals. Beyond this point, the average fitness fluctuates within this range, occasionally exceeding 400. This apparent plateau occurs because the population retains a mix of high and low fitness solutions due to stochastic selection and mutation. Unlike the best solution, which steadily improves through elitism and selection pressure, the population as a whole maintains diversity to avoid premature convergence, resulting in oscillations in the average fitness.
This behavior might seem counter-intuitive, but it actually highlights a powerful property of genetic algorithms: the best solution can improve independently of the population's average quality. This emphasizes the crucial balance between exploration and exploitation in evolutionary search.

5.3 Portfolio Optimization Problem

Portfolio optimization is a fundamental problem in finance, where the objective is to allocate a limited amount of capital among a set of assets in a way that balances expected return against risk. Investors aim to construct a portfolio that meets a desired return while minimizing risk, often measured as the variance or standard deviation of portfolio returns. In this context, constraints such as the total capital (weights summing to 1) and limits on individual asset allocations must be satisfied.

Implementation

In the GA implementation, each candidate solution is represented as a vector of asset weights, where each element corresponds to the proportion of total capital invested in a particular asset. The sum of the vector should ideally be 1, and each weight must remain within the range [0,1]. This representation directly encodes feasible portfolios and allows the GA to explore the search space through genetic operators such as crossover and mutation.
The methodology follows a standard GA workflow but includes additional mechanisms to handle constraints. Initially, a population of candidate portfolios is generated randomly. For each individual, the fitness function evaluates its risk based on the portfolio variance calculated from the covariance matrix of asset returns. Penalty terms are added to the fitness function to account for violations of the target return, sum of weights, or weight bounds, ensuring that infeasible portfolios are penalized.
In this GA implementation, each candidate solution (chromosome) is represented as a vector of asset weights, corresponding to the proportion of total capital allocated to each asset. For a portfolio of n assets, a chromosome is an n-dimensional vector [w1, w2, ..., wn] with each $w_i \in [0, 1]$. This representation allows the algorithm to directly model feasible portfolios and apply genetic operators that respect the structure of the problem.
The algorithm begins with the initialization of a population of random portfolios. Each individual in this population is generated by assigning random values to each asset weight, providing a diverse starting point for exploration. The fitness of each portfolio is then evaluated by calculating its total risk, determined as the quadratic form of the

covariance matrix with the asset weights. To ensure feasibility, portfolios that violate constraints are penalized: deviations from the total weight constraint, failure to meet a target return, or any asset weights outside the permissible range incur additional penalty terms. These penalties effectively steer the GA away from infeasible regions of the solution space and guide the search toward high-quality portfolios.
Once fitness values are assigned, the algorithm proceeds with the selection of parents for the next generation. In our implementation, tournament selection is used, in which small groups of individuals are randomly sampled from the population and the best-performing individual in each group is chosen as a parent. This method favors better solutions while preserving genetic diversity. Offspring are then generated using a two-point crossover operator, which combines segments of two parent portfolios to produce two new portfolios. This operator enables the algorithm to inherit favorable traits from multiple parents, promoting the exploration of promising regions in the solution space. Mutation is applied to each offspring to introduce variability and prevent premature convergence. In this context, mutation consists of adding Gaussian noise to individual asset weights with a certain probability, slightly adjusting the allocations and allowing the algorithm to refine the solutions incrementally. Additionally, the GA maintains a Hall of Fame of elite solutions, ensuring that the best portfolios discovered in previous generations are preserved and carried forward. This elitism mechanism guarantees that high-quality solutions are not lost due to stochastic effects in selection or recombination. To further enhance exploration, the algorithm incorporates a restart mechanism. At regular intervals, a portion of the population is reinitialized randomly while preserving the elite solutions from the Hall of Fame. Simultaneously, the crossover and mutation rates are perturbed randomly within predefined ranges. This adaptive restart strategy maintains diversity in the population and enables the GA to escape potential local optima. The process of evaluation, selection, recombination, mutation, and population update is repeated for a predefined number of generations. The algorithm continuously tracks the best solution found across all generations, which is ultimately returned as the optimal portfolio.

Results

Two instances of the portfolio optimization problem were solved using the GA, each containing three assets with differing expected returns and covariances.

Instance	Asset 1	Asset 2	Asset 3	Return	Risk
rpop_data_1	61.8%	23.1%	15.9%	1.1501	0.019831
rpop_data_2	46.7%	3.0%	51.0%	0.0500	0.022827

Table 2: Portfolio optimization results using Genetic Algorithms with 1,000 generations.

The results are summarized in Table 2. Focusing on instance 'rpop_data_1', the GA produced a portfolio with 61.8% allocated to Asset 1, 23.1% to Asset 2, and 15.9% to Asset 3. This portfolio achieved a return of 1.1501 and a risk of 0.019831. In contrast, for 'rpop_data_2', the algorithm generated a solution allocating 46.7% to Asset 1, 3.0% to Asset 2, and 51.0% to Asset 3, resulting in a return of 0.0500 and a slightly higher

risk of 0.022827. These results indicate that the GA successfully identified feasible portfolios that respect the return and weight constraints while providing a balance between expected return and risk.
Comparing these results with the known optimal solutions provides insight into the performance of the algorithm. For 'rpop_data_1', the optimal allocation is 53.01% to Asset 1, 35.64% to Asset 2, and 11.35% to Asset 3. The GA solution overweights Asset 1 and underweights Asset 2, while slightly overweighting Asset 3. This difference explains the slight deviation in return and risk: the GA solution achieves a slightly higher return (1.1501 vs. the theoretical optimum), but the risk is also a little higher due to the concentration in Asset 1. Despite this, the solution remains close to optimal. For 'rpop_data_2', the optimal solution is 49.70% to Asset 1, 0% to Asset 2, and 50.30% to Asset 3. The GA portfolio slightly underweights Asset 1, slightly overweights Asset 3, and retains a small allocation to Asset 2 (3%), which theoretically should be zero. Despite this difference the return for this solution and for the optimal solution is the same (0.0500). However, the optimal allocation gets a much lower risk than the current solution. The presence of a non-zero allocation to Asset 2 indicates that the stochastic nature of GA because of that GA often produce a feasible but not perfectly optimal solution.
Overall, the GA demonstrates strong performance in approximating the optimal portfolios. In both cases, the algorithm converged to solutions that are practically close to the theoretical optimum in terms of both return and risk.

5.4 Travelling Salesman Problem

The Traveling Salesman Problem (TSP) consists of finding the shortest possible route that visits a given set of cities exactly once and returns to the starting point. Each city is associated with a set of coordinates, and the distances between cities are typically measured using the Euclidean distance. The TSP is known to be NP-hard, meaning that the computational complexity increases rapidly as the number of cities grows.

Implementation

In our implementation for the TSP using GA, each candidate solution is represented as a permutation of integers, where each integer corresponds to a specific city in the instance. This permutation, or chromosome, directly encodes the order in which the cities are visited. The fitness of a given solution is defined as the total distance of the route, including the return from the last city back to the first. Since the objective is to minimize travel distance, the GA seeks solutions with the lowest possible fitness values. The GA process begins with the creation of an initial population of individuals, each representing a feasible tour generated as a random permutation of city indices. The algorithm then iteratively evolves the population through a series of generations. Selection is performed using tournament selection, in which a small subset of individuals is chosen at random and the one with the best fitness is selected as a parent. This method favors higher-quality solutions while maintaining genetic diversity in the population.

Offspring are generated from selected parents through crossover, specifically the ordered crossover operator. This operator ensures that offspring inherit city sequences from both parents while preserving a valid permutation without repeated cities. Mutation is applied using a shuffle mutation operator, which randomly swaps the positions of cities in the tour with a low probability. Mutation introduces variability into the population and helps the algorithm escape local optima by exploring previously unvisited regions of the solution space.
Elitism is incorporated into the algorithm to preserve the best-performing solutions across generations. A predefined number of top individuals are carried over unchanged to the next generation, guaranteeing that high-quality solutions are not lost during the stochastic processes of selection, crossover, and mutation.

Results

In order to evaluate the performance of the Genetic Algorithm for the Traveling Salesman Problem, we employed several widely recognized benchmark instances that are commonly used in the literature. Four instances with varying numbers of cities were selected: berlin52 (52 cities), bier127 (127 cities), pr76 (76 cities) and fl417 (417 cities). These instances allow us to assess both the solution quality and the computational efficiency of the algorithm across problems of different scales.
The GA parameters were kept consistent across all instances, with a population of 300 individuals, a maximum of 200 generations, a crossover probability of 0.9, a mutation probability of 0.1, tournament selection with size 2, and elitism preserving the top 30 individuals in each generation. Each individual in the population represents a candidate tour as a permutation of city indices, and the fitness is computed as the total Euclidean distance of the tour.

Instance	Cities	Best Distance (GA)	Optimal Distance	Elapsed Time (s)
berlin52	52	8742.84	7542	0.45
pr76	76	164782.46	108159	0.63
bier127	127	241985.24	123475	1.02
fl417	417	217156.70	171379	14.73

Table 3: Summary of TSP results using Genetic Algorithms with 200 generations.

From the results, it is evident that the GA is able to find high-quality solutions that are reasonably close to the known optima. While there is a gap between the GA best distances and the literature-optimal distances, this is expected given the stochastic nature of the algorithm and the limited number of generations. The quality of solutions remains impressive, particularly considering that no domain-specific heuristics were incorporated beyond basic GA operators.
An important observation is the computational efficiency. The total execution time increases with the number of cities, as expected, but not in a strictly linear manner. For berlin52 the algorithm required less than a second, while for pr76 it required approximately 0.63 seconds. For bier127, which has more than 100 cities, the execution time is slightly above 1 second, and for the largest instance, fl417, the GA completes

in just over 14 seconds. This demonstrates that, even as the number of cities grows substantially, the GA maintains reasonable scalability and efficiency, making it suitable for medium-to-large TSP instances without excessive computational burden.

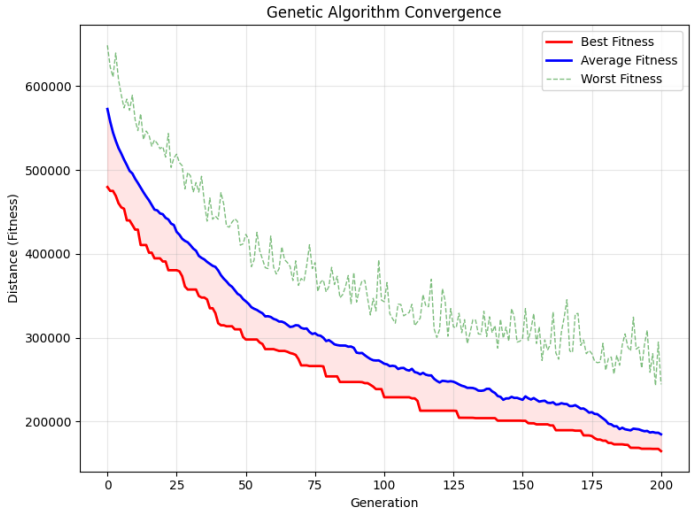


Figure 15: Convergence plot of the Genetic Algorithm for the pr76 instance.

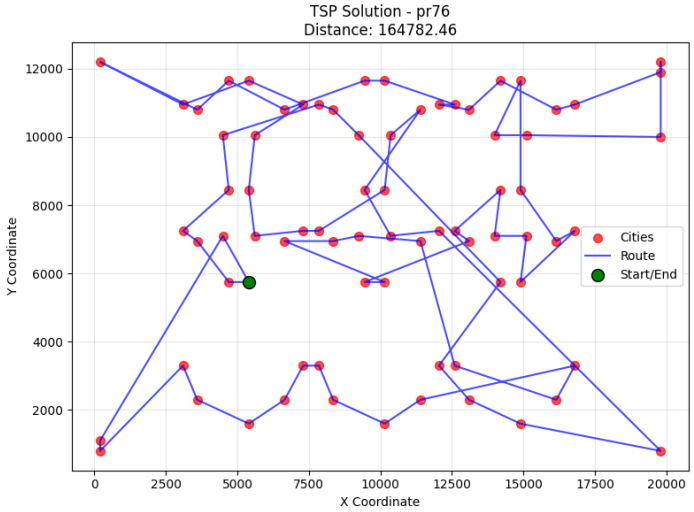


Figure 16: Visual representation of the best TSP tour obtained for the pr76 instance using the Genetic Algorithm.

Focusing on the pr76 instance, which presents an intermediate problem size, the GA required approximately 0.63 seconds to complete 200 generations. When observing the convergence behavior (Figure 15), the best fitness decreases steadily over the generations, starting from slightly below 500,000 and reaching approximately 165,000 by the final generation. Despite certain regions of stagnation, the trend shows consistent improvement. Similarly, both the worst fitness and the average fitness per generation exhibit a decreasing pattern, although their values remain higher than the best fitness,

reflecting the variability in the population. Examining the final solution, the route visualization (Figure 16) indicates that while the path could potentially be further optimized, it demonstrates a coherent and consistent traversal of the city locations.

5.5 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) objective is to determine a set of routes for a fleet of vehicles originating from a central depot, such that a set of geographically dispersed customers is served while minimizing the total operational cost. Each vehicle has a limited carrying capacity, and each customer has a specific demand that must be satisfied. Additional constraints may include time windows, route length limits, and heterogeneous fleets.

Implementation

In our implementation the solutions are represented as sequences of customer for each vehicle, a list of lists where each sub-list represents a vehicle. Each route begins and ends at the depot, and customers are assigned to a route in such a way that the vehicle's capacity is not exceeded. This representation allows the algorithm to manipulate the complete set of vehicle routes while ensuring that the assignment of customers to vehicles can be easily evaluated and updated.

1. **Fitness Evaluation and Penalties:** The fitness of a solution is defined as the total Euclidean distance of all generated routes, including the travel from and back to the central depot. To maintain the robustness of the search, a Penalty Factor (1000.0) is multiplied by any excess demand if a solution somehow violates feasibility, though the splitting mechanism primarily ensures feasibility.

2. **Selection:** The GA employs Tournament Selection with a size of $k = 3$. This mechanism selects three individuals at random and chooses the one with the lowest total distance to act as a parent. This maintains a balance between choosing high-quality parents and preserving enough diversity to avoid premature convergence.

3. **Crossover (Order Crossover - OX):** To generate offspring, the algorithm uses the Order Crossover operator. This is a specialized operator for permutations that preserves the relative order of elements from Parent 1 while filling the remaining positions with elements from Parent 2 in their order of appearance. This ensures that the offspring represent valid tours without duplicate or missing customers.

4. **Mutation and Repair:** The Swap Mutation operator is used with a probability of 0.2. It randomly selects two positions in the tour and swaps the customers. Following crossover and mutation, a Repair Function is executed to guarantee that the chromosome remains a valid permutation of all customer IDs, ensuring no customer is omitted or visited twice.

5. **Elitism and Evolution:** To ensure that the best-found routes are never lost, an Elitism strategy preserves the top 5 individuals of each generation. The population of 100 individuals is evolved over 300 generations, continuously refining the sequence of visits to reduce the cumulative distance.

Results

The VRP instance P-n70-k10 corresponds to a well-known benchmark with 69 customers plus a depot, a vehicle capacity of 135 units, and a total demand of 1313 units. Given these characteristics, the theoretical minimum number of vehicles required is 10, which already indicates a relatively tight capacity setting and a non-trivial routing structure.
The genetic algorithm successfully finds a feasible solution with a total distance of 1053.30, no penalty terms, and 11 routes, which is only one vehicle above the theoretical minimum. This result is relevant, as achieving feasibility in capacitated VRP instances is often more challenging than minimizing distance alone. The absence of penalties confirms that all capacity constraints are respected. Moreover, the execution time of 2.54 seconds highlights the computational efficiency of the approach, especially considering the combinatorial complexity of a 70-node VRP. Overall, the numerical results indicate that the GA is able to quickly converge to a high-quality and feasible solution for a medium-sized benchmark instance.

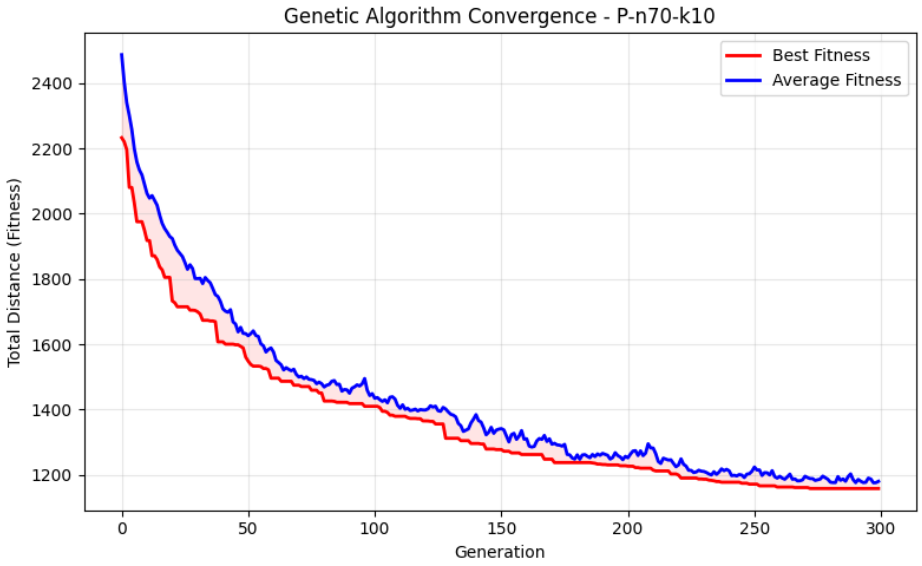


Figure 17: Convergence behavior of the genetic algorithm for the VRP instance P-n70-k10.

Figure 17 illustrates the evolution of the best fitness and average fitness values across generations. At the beginning of the optimization process, both curves show a steep decrease, reflecting the algorithm's ability to rapidly exploit low-hanging improvements in the solution space. This phase corresponds to the early generations, where crossover and mutation operators recombine diverse individuals and significantly reduce the total routing distance.

As the number of generations increases, the slope of the best fitness curve gradually flattens, indicating a transition from exploration to exploitation. Improvements still occur, but at a slower rate, suggesting that the algorithm is refining existing high-quality solutions rather than discovering radically new ones. The average fitness follows a similar trend, remaining consistently above the best fitness, which reflects the diversity preserved in the population. The narrowing gap between the two curves over time indicates a reduction in population variance and progressive convergence toward a stable region of the search space.

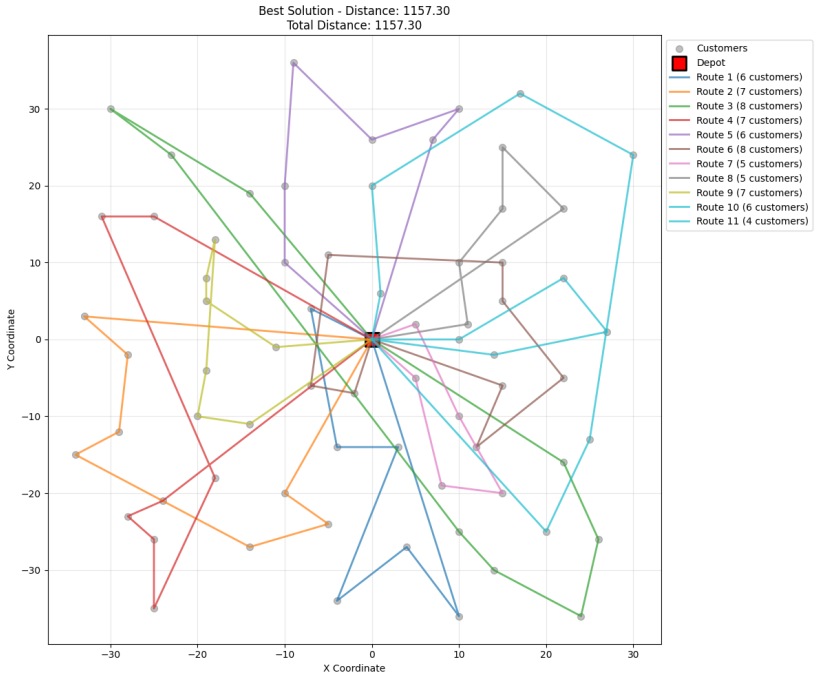


Figure 18: Routing solution obtained with the genetic algorithm for the VRP instance P-n70-k10.

As observed in Figure 18, which depicts the final set of routes for the instance, some routes may still appear visually susceptible to further improvement, particularly in terms of minor detours or boundary crossings. However, despite this, the overall routing structure exhibits a clear geographical coherence. Customers that are spatially close tend to be grouped within the same routes, and the routes themselves form compact clusters around the depot. This behavior is consistent with good VRP solutions and suggests that the GA effectively captures the spatial structure of the instance, even without explicit clustering mechanisms.

Impact of Different Crossover Operator

To further analyze the behavior of the algorithm, an additional experiment was conducted to assess the impact of different crossover operators, namely Order Crossover (OX), Partially Matched Crossover (PMX), and Cycle Crossover (CX).
As shown in Figure 19, OX initially achieves faster improvements, obtaining better solutions in the early generations compared to PMX and CX. This indicates a stronger

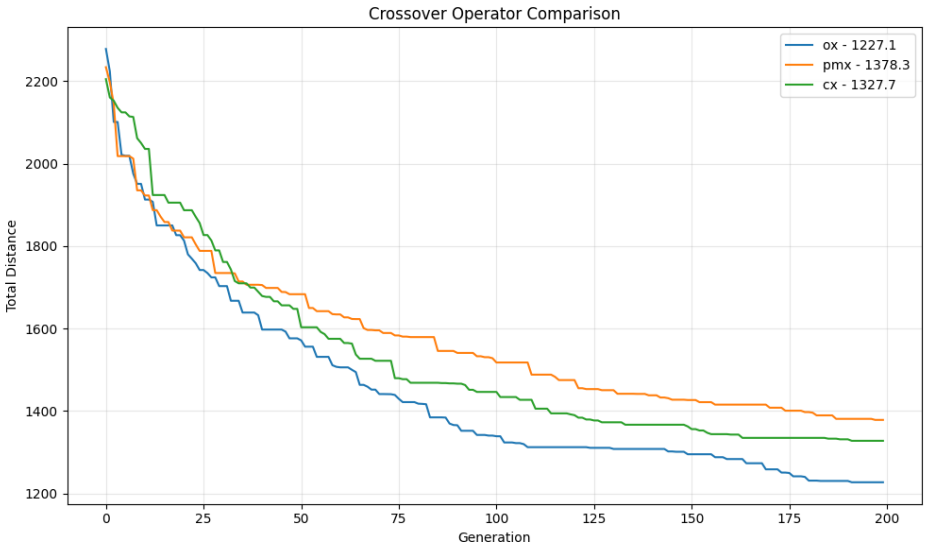


Figure 19: Comparison of crossover operators for the VRP instance P-n70-k10.

exploratory capability at the beginning of the search. However, as the number of generations increases, this advantage diminishes. CX progressively closes the gap and stabilizes at competitive fitness levels, while PMX shows a slower but more conservative improvement pattern. Ultimately, OX achieves the best final solution among the three operators, while CX reaches an intermediate performance and PMX converges to a higher final distance.

These results highlight that crossover choice has a significant impact not only on convergence speed but also on final solution quality, reinforcing the importance of operator selection when designing GA-based approaches for VRP.

References

Kizys, R., Juan, A., Sawik, B., & Calvet, L. (2019). A biased-randomized iterated local search algorithm for rich portfolio optimization. *Applied Science (Basel)*, *9*(17), 3509.

Neroni, M., Bertolini, M., & Juan, A. A. (2024). A biased-randomized discrete event algorithm to improve the productivity of automated storage and retrieval systems in the steel industry. *Algorithms*, *17*(1), 46.

Guimarans, D., Dominguez, O., Panadero, J., & Juan, A. A. (2018). A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. *Simulation Modelling Practice and Theory*, *89*, 1–14.

Dominguez, O., Juan, A. A., & Faulin, J. (2014). A biased-randomized algorithm for the two-dimensional vehicle routing problem with and without item rotations. *International Transactions in Operational Research*, *21*(3), 375–398.

White, K. P., Jr., & Ingalls, R. G. (2015). Introduction to simulation. *Proceedings of the 2015 Winter Simulation Conference*.

Raychaudhuri, S. (2008). Introduction to monte carlo simulation. *Proceedings of the 2008 Winter Simulation Conference*, 91.

Biller, B., & Nelson, B. L. (2002). Answers to the top ten input modeling questions. In E. Yücesan, C.-H. Chen, J. L. Snowdon, & J. M. Charnes (Eds.), *Proceedings of the 2002 winter simulation conference*. IEEE Press.

Raba, D., Juan, A. A., Panadero, J., Bayliss, C., & Estrada-Moreno, A. (2019). Combining the internet of things with simulation-based optimization to enhance logistics in an agri-food supply chain. *Proceedings of the 2019 Winter Simulation Conference*.

Onggo, B. S., Juan, A. A., Panadero, J., Corlu, C. G., & Agustin, A. (2019). An inventory-routing problem with stochastic demand and stock-out: A solution and risk analysis using simheuristics. *Proceedings of the 2019 Winter Simulation Conference*.

Calvet, L., Ferrer, A., Gomes, M. I., Juan, A. A., & Masip, D. (2016). Combining statistical learning with metaheuristics for the Multi-Depot vehicle routing problem with market segmentation. *Computers & Industrial Engineering*, *94*, 93–104.

Fikar, C., Juan, A. A., Martinez, E., & Hirsch, P. (2016). A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing. *European Journal of Industrial Engineering*, *10*(3), 323–340.

Martins, L. d. C., Tarchi, D., Juan, A. A., & Fusco, A. (2022). Agile optimization for a real-time facility location problem in internet of vehicles networks. *Networks*, *79*(4), 501–514. https://doi.org/10.1002/net.22067