# VRPAgent: LLM-Driven Discovery of Heuristic Operators for Vehicle Routing Problems

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Designing high-performing heuristics for vehicle routing problems (VRPs) is a complex task that requires both intuition and deep domain knowledge. Large language model (LLM)-based code generation has recently shown promise across many domains, but it still falls short of producing heuristics that rival those crafted by human experts. In this paper, we propose VRPAgent, a framework that integrates LLM-generated components into a metaheuristic and refines them through a novel genetic search. By using the LLM to generate problem-specific operators, embedded within a generic metaheuristic framework, VRPAgent keeps tasks manageable, guarantees correctness, and still enables the discovery of novel and powerful strategies. Across multiple problems, including the capacitated VRP, the VRP with time windows, and the prize-collecting VRP, our method discovers heuristic operators that outperform handcrafted methods and recent learning-based approaches while requiring only a single CPU core. To our knowledge, VRPAgent is one the first LLM-based methods to advance the state-of-the-art in VRPs, highlighting a promising future for automated heuristics discovery.

## 1 Introduction

Solving combinatorial optimization problems requires sophisticated solution approaches. This is especially true for vehicle routing problems (VRPs), where real-world instances often involve complex constraints and a large number of customers. Over the past decades, operations researchers have developed countless heuristics to address these problems (Konstantakopoulos et al., 2022). Designing a new method that meaningfully improves upon the state of the art across multiple problems is extremely challenging, requiring years of experience and a deep understanding of both general heuristics and the specific problem at hand. Practitioners face similar challenges when applying heuristics. Real-world applications often involve ever-changing requirements that are not supported by existing solution methods. Adapting approaches from the literature to such requirements is a time-consuming and challenging task, and is often considered impractical, even by large companies.

In recent years, neural combinatorial optimization (NCO) has garnered increasing attention due to its potential to automate the discovery of effective heuristics (Bello et al., 2017; Bengio et al., 2021). NCO approaches aim to solve optimization problems by training deep neural networks, typically with reinforcement learning. While NCO methods have demonstrated the ability to learn powerful solution strategies for various combinatorial problems, they also come with notable limitations. First, they require expensive GPUs at test time, which restricts their practical deployment. Second, scalability remains a major challenge as their reliance on attention mechanisms makes it difficult to apply these models to problems that involve processing full distance matrices. Finally, the learned strategies are often difficult for experts to interpret, which raises concerns about their safety and reliability in real-world applications.

The recent advent of performant large language models (LLMs) has enabled new opportunities for automation in general algorithmic design across domains ranging from code synthesis to symbolic planning and mathematical discovery (Madaan et al., 2023). LLMs have proven to be promising approach for discovering new heuristics in combinatorial optimization problems: they can be used to design new heuristics from scratch or adapt existing ones to real-world requirements, enabling customized solutions at a fraction of the cost of an

operations research (OR) expert. Among pioneering works, Romera-Paredes et al. (2024); Liu et al. (2024a); Ye et al. (2024a) propose evolutionary frameworks that iteratively evolve general CO problem heuristics. Recent research has focused on increasingly sophisticated approaches for automating heuristic discovery (Dat et al., 2025; Zheng et al., 2025; Yang et al., 2025b; Novikov et al., 2025; Liu et al., 2025). Although these works provide valuable contributions, the discovered heuristics still fall short of those designed by human experts for VRPs. We identify key limitations in most of these works: design of end-to-end functions, absence of correctness guardrails and overall solution frameworks, and inefficient exploration of the search space, which leads to a failure in challenging the state-of-the-art.

We introduce VRPAgent, a novel approach that uses LLMs to design heuristic operators for a large neighborhood search (LNS). The high-level LNS is designed to be largely problem-agnostic, allowing our framework to tackle new problems by creating new heuristic operators with minimal human input. To discover strong operators for the LNS, we employ a genetic algorithm (GA) with elitism and biased crossover that iteratively improves operator quality. By focusing on a metaheuristic framework where only problem-specific operators are generated via LLMs, we keep the generation task manageable and effective, while still enabling strong performance on complex problems. We evaluate our method on the capacitated vehicle routing problem (CVRP), the vehicle routing problem with time windows (VRPTW), and the prize-collecting VRP (PCVRP). Our approach discovers heuristic operators for all problems that significantly outperform those designed by human experts.

In summary, we make the following contributions with VRPAgent:

- We propose a novel framework in which LLM generation of problem-specific operators is constrained in a correctness-enforcing LNS-based metaheuristic.
- We introduce a simple GA for heuristic discovery featuring elitism with biased crossover for improved exploitation, and a code length penalty to reduce LLM inference costs.
- We show that VRPAgent discovers strong heuristics across multiple tasks. To the best of our knowledge, it is one of the first LLM-based approaches to advance the state-of-the-art in VRPs.

## 2 Related Work

**Traditional Heuristics** VRPs are ubiquitous problems in logistics that have been studied for decades. Large and richly constrained instances remain difficult to solve to optimality within a practical time frame, and thus heuristics are commonly used in real-world settings (Santini et al., 2023). LNS and its adaptive variants are particularly influential, iteratively destroying and repairing parts of a solution (Shaw, 1998; Schrimpf et al., 2000; Christiaens & Vanden Berghe, 2020). Other well-known approaches include LKH3 (Helsgaun, 2017) and hybrid genetic search (HGS) (Vidal, 2022; Wouda et al., 2024). While capable of producing high-quality solutions, these approaches take significant expertise to design and implement, motivating the need for automating their design.

**Neural Combinatorial Optimization** NCO aims to automate heuristic design by training neural networks from data or via reinforcement learning (Bengio et al., 2021; Berto et al., 2025a; Li et al., 2025b). Approaches can be broadly divided into construction and improvement methods. Construction methods, such as pointer networks (Vinyals et al., 2015; Bello et al., 2017) and subsequent attention-based models for VRPs (Kool et al., 2019; Kwon et al., 2020; Kim et al., 2022; Berto et al., 2025b; Huang et al., 2025a), generate complete solutions in an autoregressive fashion. Further works include enhancements for diversity (Grinsztajn et al., 2023; Hottung et al., 2025a) and out-of-distribution robustness (Drakulic et al., 2023; Luo et al., 2023). Improvement methods instead refine existing solutions at test time, for example, by learning local edits (Ma et al., 2021), guiding $k$-opt moves (Wu et al., 2019; da Costa et al., 2020; Ma et al., 2023), or integrating with metaheuristic approaches as LNS (Hottung & Tierney, 2020) or ant colony optimization (Ye et al., 2023; Kim et al., 2025). Divide-and-conquer frameworks further extend scalability to large instances (Kim et al., 2021; Li et al., 2021; Ye et al., 2024b; Ouyang et al., 2025). Hottung et al. (2025b) adopts a LNS approach with learned heuristics for deconstruction and ordering VRP nodes, showing competitive results against

state-of-the-art solvers. Despite continuous progress, most NCO work still falls short of handcrafted solvers and requires expensive GPU resources, motivating our use of LLM-discovered operators as a lightweight alternative.

**Automated Heuristic Discovery** The goal of automatically discovering high-performing heuristics is a long-standing challenge in optimization (Muth, 1963). Early works include genetic programming and hyper-heuristics, which construct new solution methods by combining or tuning a set of low-level heuristic components (Burke et al., 2006; 2013) and grammar-based generation (Mascia et al., 2014). The recent advent of LLMs has enabled a new wave of automation for algorithmic design across domains ranging from code synthesis to symbolic planning and mathematical discovery (Madaan et al., 2023; Shinn et al., 2023; Novikov et al., 2025). Early works in heuristic discovery with LLMs including Romera-Paredes et al. (2024); Liu et al. (2024a) employ evolutionary approaches that generate heuristic code snippets for simple heuristics in combinatorial problems, including VRPs, packing, and scheduling. Building on this trend, reflection-augmented evolution has been shown to discover more sophisticated heuristics during the refinement process (Ye et al., 2024a). Orthogonal search strategies further expand the design space: diversity-driven evolution (Dat et al., 2025), Monte Carlo tree search (Zheng et al., 2025), ensembling of different LLMs (Novikov et al., 2025), meta-prompt optimization (Shi et al., 2025), and portfolio-style discovery of sets of complementary heuristics (Yang et al., 2025b; Liu et al., 2025). More specifically for VRPs, Tran et al. (2025) design heuristics to enhance NCO model decoding. In parallel, finetuning and instruction specialization of LLMs for algorithmic synthesis have been proposed to improve reliability and sample efficiency (Šurina et al., 2025; Huang et al., 2025b; Chen et al., 2025b), and benchmark suites have begun to standardize evaluation protocols for LLM-driven heuristics (Liu et al., 2024b; Sun et al., 2025; Feng et al., 2025; Li et al., 2025a; Chen et al., 2025a).

Despite encouraging progress, LLM-generated heuristics still lag behind traditional solvers and NCO methods alike on VRPs, especially under tight time budgets and realistic constraints. We identify three recurring limitations: (i) weak "agentic playground" formulations that ask LLMs to design small heuristic snippets without an overall solution framework; (ii) weak or absent correctness guards around generated code; and (iii) inefficient exploration that drifts toward verbose, brittle implementations. Our approach follows the principle of *keeping AI agents on a leash*: we constrain the search to problem-specific operators nested within a robust, correctness-enforcing metaheuristic. VRPAgent's design keeps the synthesis task tractable, preserves feasibility, and still enables the discovery of novel operators that can advance the state-of-the-art for VRP solving.

## 3 Vehicle Routing Problems

VRPs are a fundamental class of combinatorial optimization problems with the aim to minimize travel costs while respecting some constraints. Travel cost is usually measured by the total distance traveled. Formally, a VRP is defined on a graph $G = (V, E)$, where each node $i \in V$ denotes a customer and each edge $(i, j) \in E$ models traveling from $i$ to $j$ with an associated cost, e.g., the distance between $i$ and $j$. All routes originate from and end at the depot node 0. In the **CVRP**, vehicles performing the routes have a limited capacity. The total demand on any route cannot exceed the vehicle's capacity $C$ at any time, and every customer is served exactly once. The **VRPTW** extends this setting, assigning a service time $s_i$ and a time window $[t_i^l, t_i^r]$ to each customer $i$. The service to any customer must start within their time window, i.e., if a vehicle arrives before a customer's time window starts, it has to wait until $t_i^l$. The **PCVRP** relaxes the requirement of visiting all customers. Servicing a customer $i$ is associated with a prize $p_i$. The objective is to maximize the total collected prize while minimizing travel cost.

## 4 VRPAgent

VRPAgent is a framework for solving VRPs that automatically discovers strong heuristic operators using LLMs. It is built on two main components. The first (Section 4.1) is an LNS (Shaw, 1998) variant for VRPs, which relies on heuristic operators to improve solutions iteratively. At test time, this LNS produces solutions for VRP instances on a single CPU core. The second component (Section 4.2) is a GA used in a discovery

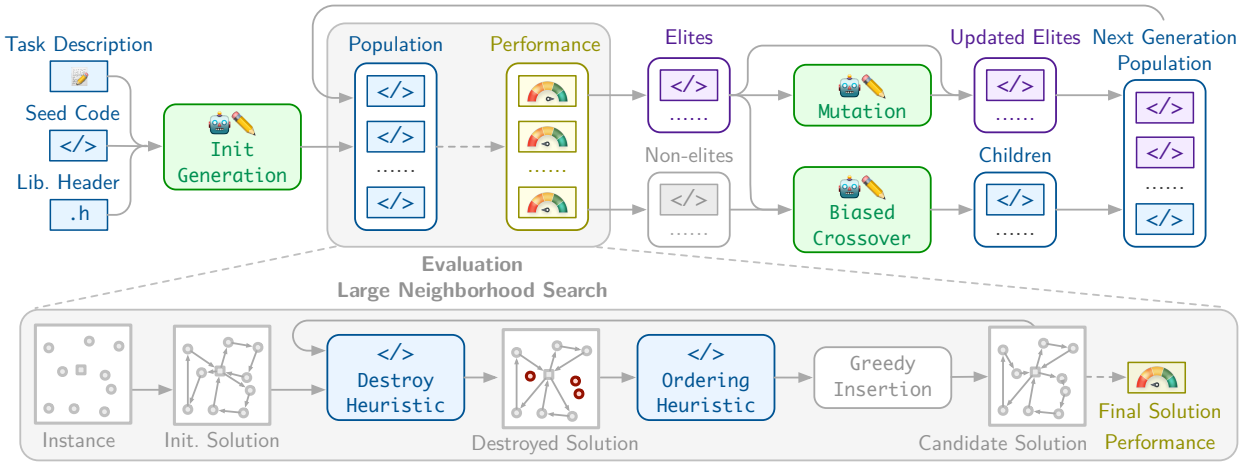



Figure 1: VRPAgent overview.

phase to generate these heuristic operators. In the discovery phase, operator implementations are iteratively created, modified, and refined with the help of an LLM. Classic genetic operations such as crossover and mutation are applied to operator implementations, with the LLM carrying out these transformations. Each generated operator is evaluated by inserting it into the LNS and testing the resulting search performance on a set of training instances. The performance on these instances defines the fitness value of the individual. Over successive generations, the GA produces increasingly effective heuristic operators. Fig. 1 shows a high-level overview of VRPAgent.

### 4.1 Large Neighborhood Search with LLM-generated Operators

VRPAgent employs an LNS with LLM-generated operators to find solutions for VRPs. The high-level LNS guides the search process and ensures feasibility by acting as a safeguard around the LLM-generated code. In short, the LNS works by repeatedly removing a set of customers from their tours, ordering the removed customers, and reinserting them one by one at their locally optimal positions. The removal and ordering strategies are defined by LLM-written heuristic operators.

Algorithm 1 presents the pseudocode of our LNS. The algorithm begins by generating an initial solution $s$ for a given instance $l$. For all routing problems, this initial solution is constructed with one tour per customer. The solution is then iteratively improved until a termination criterion is met. In each iteration, $s$ is first destroyed by removing customers from their tours using the LLM-generated removal operator $f_{\text{Remove}}$. This yields an incomplete solution $s'$ in which the removed customers are unassigned. Next, the removed customers are ordered by the LLM-generated operator $f_{\text{Order}}$. They are then reinserted one by one in that order, always placed at the locally best position. That is, the insertion that increases the objective value as little as possible. Finally, an acceptance decision is made: $s'$ may replace $s$ only if it is better, or it may be accepted under a simulated annealing rule. After all iterations, the algorithm returns the best solution $s$.

### 4.2 Heuristic Discovery

VRPAgent discovers heuristic operators using a simple GA that is strongly geared toward exploitation. Given the very large search space and limited search budget, this bias toward exploitation proves highly beneficial, leading to significant improvements in our experiment. Each individual in our GA represents an implementation of the operator pair $(f_{\text{Remove}}, f_{\text{Order}})$ as C++ code. During the discovery phase, new individuals are created through the means of mutation and crossover. To evaluate an individual, we run VRPAgent-LNS using its operator pair on a set of training instances.

Algorithm 2 outlines the core logic of our GA. It takes as input the initial population size $M_{\text{init}}$, the number of elites $M_{\text{E}}$, and the number of offspring $M_{\text{C}}$ generated in each iteration. The algorithm begins by creating

**Algorithm 1** VRPAgent-LNS

**Input:** CVRP Instance $l$, Destroy Operator $f_{\text{Remove}}$, Ordering Operator $f_{\text{Order}}$

$s \leftarrow \text{GenerateStartSolution}(l)$
**while** termination criteria not reached **do**
  $s' \leftarrow f_{\text{Remove}}(l, s)$
  $insertionOrder \leftarrow f_{\text{Order}}(l, s')$
  **for** $c$ in $insertionOrder$ **do**
    Insert customer $c$ at their locally optimal position in $s'$
  **end for**
  $s \leftarrow \text{Accept}(s, s')$
**end while**
**return** $s$

an initial population (i.e., a set) of heuristic operators and then enters the main evolutionary loop, which runs until a termination criterion is reached. At the start of each iteration, all individuals are evaluated, and the top $M_{\text{E}}$ are placed in the set of elites $\mathcal{P}_E$, and the reminder in the set of non-elites $\mathcal{P}_{NE}$. Next, $M_{\text{C}}$ offspring are created by pairing one elite with one non-elite individual and combining them using biased crossover. This crossover favors the elite parent while still injecting diversity from the non-elite.

**Algorithm 2** VRPAgent-GA

**Input:** Initial population size $M_{\text{init}}$, number of elites $M_{\text{E}}$, number of offspring $M_{\text{C}}$

$\mathcal{P} \leftarrow \text{GenerateStartPop}(M_{\text{init}})$
**while** termination criteria not reached **do**
  $(\mathcal{P}_E, \mathcal{P}_{NE}) \leftarrow \text{Top-K-Elite}(\mathcal{P}, M_{\text{E}})$
  $C \leftarrow \emptyset$
  **while** $|C| < M_{\text{C}}$ **do**
    $p_e \leftarrow \text{Random}(\mathcal{P}_E)$
    $p_{ne} \leftarrow \text{Random}(\mathcal{P}_{NE})$
    $p_c \leftarrow \text{Biased-Crossover}(p_e, p_{ne})$
    $C \leftarrow C \cup \{p_c\}$
  **end while**
  **for all** $p_e \in \mathcal{P}_E$ **do**
    $p_m \leftarrow \text{Mutation}(p_e)$
    **if** $\text{Fit}(p_m) < \text{Fit}(p_e)$ **then**
      $p_e \leftarrow p_m$
    **end if**
  **end for**
  $\mathcal{P} \leftarrow \mathcal{P}_E \cup C$
**end while**
**return** $\text{best}(\mathcal{P})$

Each elite is refined through mutation. Unlike standard GAs, where mutation is typically applied to offspring to promote exploration, we apply it directly to elites. These mutations are small, making the procedure more exploitation-focused. If a mutated elite achieves better fitness, it replaces the original; otherwise, the original is kept. This replacement rule ensures that mutated elites do not accumulate in the population, thereby preventing premature convergence. Finally, the next generation is formed by combining the elites with the newly generated offspring. The process repeats until termination, after which the best heuristic operator discovered is returned. We describe the initialization, crossover, mutation, and fitness evaluation steps in the following paragraphs.

**Initial Population Generation** The initial population is created by prompting the LLM to generate implementations of the two operators. For this, the LLM is provided with a global system context that

explains the overall problem and the LNS, a trivial example implementation of both operators, and technical details of the LNS implementation in the form of C++ header files that allow operators to access shared variables and methods efficiently (see Prompts 1, 2 and 9 in Appendix A).

**Biased Crossover** The offspring are produced by combining two parents through crossover. We use biased crossover, which pairs an elite individual with a non-elite one. The LLM is given both their implementations and is prompted to take most ideas and concepts from the implementation of the elite individuals and only a predefined % from the implementation of the non-elite individual. This adaptation of standard crossover significantly increases exploitation. The complete crossover instruction provided to the LLM corresponds to Prompts 1 and 3 in Appendix A.

**Mutation** VRPAgent uses mutation to slightly modify elite implementations. Following Liu et al. (2024a), we implement multiple mutation prompts that focus on different areas of improvement for better exploration. More precisely, the LLM is given the implementation that should be modified together with one randomly selected mutation prompt (Prompts 4 to 7 in Appendix A) and the global system context Prompt 1. The four supported mutations include *Ablation* (i.e., removing a random mechanic), *Extend* (i.e., adding a new mechanic), *Adjust-Parameters* (i.e., changing the hyperparameter settings), and *Refactor* (i.e, modify the code so that the runtime is improved).

**Fitness Function with Code Length Penalty** We evaluate an individual $i$ by running VRPAgent-LNS with the operator pair defined by $i$ on a set of training instances $I^{\text{train}}$. The resulting solutions are then used to compute the fitness value. Specifically, the fitness of $i$ is defined as the average objective value across all training instances, plus a penalty proportional to the length (i.e., number of lines) of the corresponding implementation $\mathcal{C}_i$:

$$\text{Fit}(i) \;=\; \frac{1}{|I^{\text{train}}|} \sum_{j \in I^{\text{train}}} \text{Obj}(s_{i,j}) \;+\; \lambda \cdot \text{Len}(\mathcal{C}_i), \tag{1}$$

where $s_{i,j}$ is the solution obtained by applying VRPAgent-LNS with the operators of individual $i$ to training instance $j$, and $\lambda$ controls the strength of the code length penalty.

The penalty helps prevent uncontrolled growth in implementation size, which we observed when no regularization was applied. More compact implementations are also easier for humans to inspection and auditing, making the generated heuristics more useful in practice. Finally, a shorter code reduces the number of tokens processed by the LLM during generation, which in our experiments lowers token usage by more than 50%, and thus significantly reduces generation costs and latency.

## 5 Experiments

We evaluate VRPAgent on three vehicle routing problems: the CVRP, the VRPTW, and the PCVRP. To demonstrate its generality, we combine VRPAgent with six different LLMs, four of which are executed locally, showing that the framework does not depend on a single model. In addition, we perform ablation studies to analyze the contribution of individual components, and sensitivity analyses to investigate the effect of key parameters. The best discovered heuristics are made publicly available[1].

**VRPAgent Hyperparameters** During the discovery phase we use the following parameters unless stated otherwise: elite size $M_E = 10$, offspring size $M_C = 30$, an initial population size of $M_{\text{init}} = 100$. The code length penalty factor is set to $\lambda = 2 \cdot 10^{-4}$ and the discovery phase is terminated after 40 iterations. Individuals are evaluated with VRPAgent-LNS on a training set of 64 instances, each with 500 customers, using a runtime limit of 20s per instance. We employ Gemini 2.5 Flash (Comanici et al., 2025) as the LLM.

[1] `https://anonymous.4open.science/r/vrpagent-submission`

### 5.1 Comparison to State-of-the-Art

**Benchmark Setup** We evaluate all methods on the three problems using instance sets of 500, 1000, and 2000 customers. To ensure consistency, we adopt the same test instances and baseline configurations as described in Hottung et al. (2025b) using a single core of an AMD Milan 7763 processor and an additional single NVIDIA A100 for approaches that require a GPU. For a fair comparison, we limit the search by runtime when possible. The operators used by VRPAgent are obtained from 10 discovery runs per problem (conducted on instances of size 500 only), with the best operator selected based on performance on a separate validation set.

**Baselines** We compare VRPAgent to several established operations research solvers: HGS (Vidal, 2022), SISRs (Christiaens & Vanden Berghe, 2020), LKH3 (Helsgaun, 2017), and Gurobi (Gurobi Optimization, LLC, 2024). We also include PyVRP (Wouda et al., 2024) (version 0.9.0), an open-source extension of HGS that supports additional VRP variants, and the recent GPU-based NVIDIA cuOpt (NVIDIA Corporation, 2025). For the CVRP, we further consider learning-based approaches that require GPUs at test time: BQ (Drakulic et al., 2023), LEHD (Luo et al., 2023), UDC (Zheng et al., 2024b), and NDS (Hottung et al., 2025b). In addition, we compare against LLM-based methods that learn a construction heuristic, including EoH (Liu et al., 2024a), MCTS-AHD (Zheng et al., 2024a), and ReEvo (Ye et al., 2024a). These approaches only generate a single solution with runtime <1 second and do not benefit from additional search budget. We further compare against three stronger LLM-based baselines that leverage search: ReEvo-ACO, which combines LLM-generated heuristics with the Ant Colony Optimization (ACO) metaheuristic, NCO-LLM (Tran et al., 2025), which enhances LEHD by automating the design of logit reshaping during search, and LLM-LNS (Ye et al., 2025), a recent method using LLMs to design search heuristics.

**Results** Table 1 presents the results of our experiments. On instances with 1000 and 2000 customers, VRPAgent outperforms all other methods across all problem types, achieving gaps of around $-0.30\%$ relative to the state-of-the-art SISRs. This represents a substantial improvement that can translate into significant savings in large-scale, real-world scenarios. On smaller instances, VRPAgent approaches the performance of NDS, which relies on an expensive GPU at test time and is trained specifically for each instance size. Compared to other LLM-based methods, VRPAgent consistently demonstrates significantly better performance across all test cases. It is also noteworthy that VRPAgent performs well on all problem sizes, despite being trained only on instances with 500 customers. This strong generalization performance stems from the algorithmic operators. These usually do not rely on scale-dependent neural representations, which are poisoned to implicitly encode all characteristics of the training distribution, including unwanted characteristics like node count and spatial density.

### 5.2 Comparison on CVRPLIB Instances

We also evaluate VRPAgent on the X instance set (Uchoa et al., 2017) from CVRPLIB, which consists of 100 CVRP instances with sizes ranging from 100 to 1000 customers. This dataset is explicitly designed to encompass a wide variety of challenging real-world problem structures, including instances with highly clustered customers and heterogeneous demands. To generate a heuristic for these instances, we train VRPAgent on a separate set of 128 instances exhibiting similar characteristics. During the heuristic discovery process, the runtime for each instance is limited to $0.025 \times n$ seconds, where $n$ denotes the number of nodes.

The resulting heuristic is subsequently evaluated on the full X instance set with an increased runtime limit of $1.0 \times n$ seconds per instance. We compare the performance of the heuristic generated by VRPAgent against SISRs, the best handcrafted method from Section 5.1, under the same runtime constraints. For each instance, we perform 10 independent runs and report the average results.

Table 2 shows the results aggregated by instance size. In total, VRPAgent finds better solutions on 71 of the 100 instances, while SISRs finds better solutions on 26 instances. Notably, for larger instances with more than 500 customers, the heuristic produced by VRPAgent finds better solution on 28 out of 32 instances. The detailed per-instance results can be found in Appendix C.

Table 1: Performance on test data. The gap is calculated relative to SISRs. Runtime is reported on a per-instance basis in seconds. The best results (i.e., those with the lowest objective function value) are shown in **bold**, and the second-best are underlined. * Indicates that a feasible solution was not found for all instances. OOM indicates that the corresponding solver yielded an out-of-memory error.

| | Method | | N=500 | | | N=1000 | | | N=2000 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Obj.↓ | Gap↓ | Time | Obj.↓ | Gap↓ | Time | Obj.↓ | Gap↓ | Time |
| CVRP | SISRs | CPU | 36.65 | - | 60 | 41.14 | - | 120 | 56.04 | - | 240 |
| | HGS | CPU | 36.66 | 0.00% | 60 | 41.51 | 0.84% | 121 | 57.38 | 2.33% | 241 |
| | LKH3 | CPU | 37.25 | 1.66% | 174 | 42.16 | 2.46% | 408 | 58.12 | 3.70% | 1448 |
| | NVIDIA cuOpt | CPU+GPU | 37.38 | 1.98% | 60 | 42.71 | 3.78% | 121 | 59.22 | 5.66% | 241 |
| | Gurobi | CPU | 47.85 | 30.53% | 60 | 87.36 | 111.88% | 120 | - | OOM | - |
| | BQ (BS64) | CPU+GPU | 37.51 | 2.34% | 23 | 43.32 | 5.30% | 164 | - | - | - |
| | LEHD (RRC) | CPU+GPU | 37.04 | 1.06% | 60 | 42.47 | 3.25% | 121 | 60.11 | 7.25% | 246 |
| | UDC | CPU+GPU | 37.63 | 2.69% | 60 | 42.65 | 3.68% | 121 | - | - | - |
| | NDS | CPU+GPU | 36.57 | **-0.20%** | 60 | 41.11 | -0.07% | 120 | 56.00 | -0.07% | 240 |
| | EoH | CPU | 45.89 | 25.21% | <1 | 52.42 | 27.42% | <1 | 71.21 | 27.07% | <1 |
| | MCTS-AHD | CPU | 45.51 | 24.17% | <1 | 52.49 | 27.59% | <1 | 71.15 | 26.96% | <1 |
| | ReEvo | CPU | 44.21 | 20.63% | <1 | 52.23 | 26.96% | <1 | 70.01 | 24.93% | <1 |
| | ReEvo-ACO | CPU | 40.25 | 9.83% | 60 | 46.22 | 12.34% | 120 | 63.76 | 13.77% | 240 |
| | NCO-LLM | CPU+GPU | 36.93 | 0.76% | 60 | 41.96 | 1.99% | 121 | 59.43 | 6.05% | 246 |
| | LLM-LNS | CPU | 38.99 | 6.38% | 60 | 45.00 | 9.38% | 120 | 62.28 | 11.13% | 240 |
| | **VRPAgent** | CPU | 36.60 | -0.12% | 60 | 41.06 | **-0.19%** | 120 | 55.98 | **-0.11%** | 240 |
| VRPTW | SISRs | CPU | 48.09 | - | 60 | 87.68 | - | 120 | 167.49 | - | 240 |
| | PyVRP-HGS | CPU | 49.01 | 1.91% | 60 | 90.35 | 3.08% | 120 | 173.46 | 3.62% | 240 |
| | NVIDIA cuOpt | CPU+GPU | 49.30 | 2.60% | 61 | 90.31 | 3.11% | 121 | 173.52 | 3.85%* | 243 |
| | Gurobi | CPU | 69.01 | 43.83% | 60 | 148.45 | 71.46% | 120 | - | OOM | - |
| | NDS | CPU+GPU | 47.94 | **-0.30%** | 60 | 87.54 | -0.16% | 120 | 167.48 | -0.00% | 240 |
| | EoH | CPU | 60.40 | 25.60% | <1 | 118.80 | 35.49% | <1 | 245.70 | 46.70% | <1 |
| | MCTS-AHD | CPU | 58.31 | 21.25% | <1 | 113.72 | 29.70% | <1 | 231.11 | 37.98% | <1 |
| | ReEvo | CPU | 58.01 | 20.63% | <1 | 110.55 | 26.08% | <1 | 218.90 | 30.69% | <1 |
| | ReEvo-ACO | CPU | 52.91 | 10.03% | 60 | 97.39 | 11.07% | 120 | 193.13 | 15.31% | 240 |
| | **VRPAgent** | CPU | 47.97 | -0.24% | 60 | 87.40 | **-0.33%** | 120 | 166.96 | **-0.33%** | 240 |
| PCVRP | SISRs | CPU | 43.22 | - | 60 | 81.12 | - | 120 | 158.17 | - | 240 |
| | PyVRP-HGS | CPU | 44.97 | 4.10% | 60 | 84.91 | 4.81% | 120 | 165.56 | 4.78% | 240 |
| | NVIDIA cuOpt | CPU+GPU | 43.34 | 0.19% | 60 | 81.89 | 0.84% | 121 | 160.33 | 1.22% | 241 |
| | Gurobi | CPU | 71.58 | 68.23% | 60 | 147.08 | 86.03% | 120 | - | OOM | - |
| | NDS | CPU+GPU | 43.12 | **-0.23%** | 60 | 80.99 | -0.17% | 121 | 158.09 | -0.06% | 241 |
| | **VRPAgent** | CPU | 43.18 | -0.09% | 60 | 80.95 | **-0.21%** | 120 | 157.69 | **-0.32%** | 240 |

Table 2: Performance on the real-world-like CVRPLib X dataset. Results are averaged by customer count.

| | | Gap | | Wins | |
|---|---|---|---|---|---|
| Size $N$ | # Instances | SISRs | VRPAgent | SISRs | VRPAgent |
| 100–250 | 32 | **0.123%** | 0.134% | 12 | 17 |
| 250–500 | 36 | 0.284% | **0.215%** | 10 | 26 |
| 500–750 | 19 | 0.276% | **0.238%** | 3 | 16 |
| 750+ | 13 | 0.269% | **0.222%** | 1 | 12 |

### 5.3 Analyses

**Performance Across Different LLMs** We study the performance of VRPAgent when paired with different LLMs. We conduct discovery runs of 20 iterations each for the CVRP using six models. We access Gemini 2.0 Flash and Gemini 2.5 Flash (Comanici et al., 2025) via API, while Qwen3 (Yang et al., 2025a), Llama 3.3 (Grattafiori et al., 2024), Gemma 3 (Team et al., 2025), and gpt-oss (Agarwal et al., 2025) are served locally via vLLM (Kwon et al., 2023). Fig. 2 reports the average objective value on the training set during discovery (left) and the total computational cost per run (right). All tested models substantially

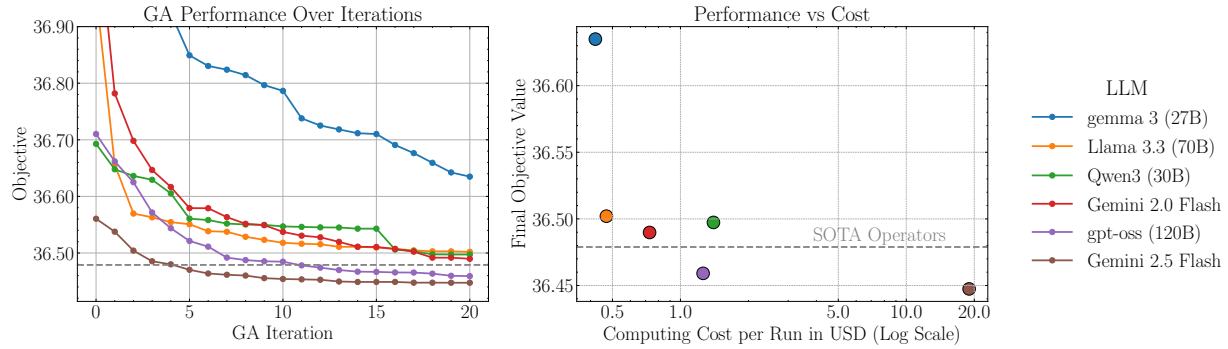



Figure 2: Performance on the CVRP for different LLMs. Detailed cost calculations are provided in Section B.1.

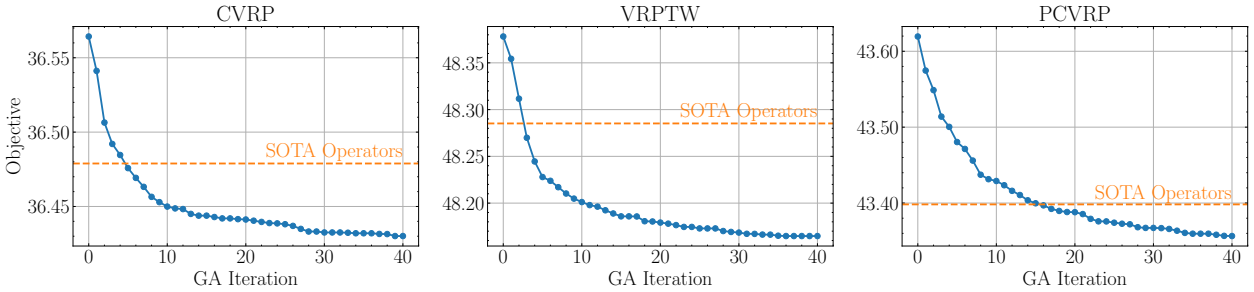



Figure 3: Performance over the course of the discovery process.

improve the heuristic operators throughout the discovery process. Gemini 2.5 Flash and gpt-oss both discover heuristics that outperform the state-of-the-art baseline. Gemini 2.5 Flash achieves the best overall results, but at a cost of nearly $20 per run. In contrast, the open-source gpt-oss model, run on two NVIDIA A100 (40GB) GPUs, achieves nearly the same performance under $2 per run.

**Performance Over the Discovery Process** We analyze the convergence rate of the discovery process with Gemini 2.5 Flash on all three problems across 40 iterations. As a baseline, we report the performance of VRPAgent-LNS when used in combination with handcrafted operators. Specifically, we reimplement the operators from SISRs (Christiaens & Vanden Berghe, 2020), which represent the state of the art in LNS-based routing methods. As shown in Fig. 3, VRPAgent produces heuristics that outperform the state-of-the-art (SOTA) handcrafted operators. In Appendix D, we additionally report the percentage of operators that compile successfully and produce valid solutions (success rate) throughout the runs.

**Ablation Studies** We analyze the contribution of key components in our GA by disabling or replacing them. Specifically, we test three variants: (i) replacing our biased crossover with a standard crossover, where the LLM is instructed to take roughly half the elements from each parent, (ii) removing mutation while increasing offspring size to maintain a comparable population, and (iii) replacing our entire GA with the GA of ReEvo (Ye et al., 2024a), which uses a reflection mechanism. Each variant is tested on the CVRP 10 times, and results are averaged. Fig. 4 reports performance on the training set during the discovery phase. Across all cases, modifications lead to reduced performance. Biased crossover is particularly important: by favoring elite solutions while still incorporating elements from weaker parents, it

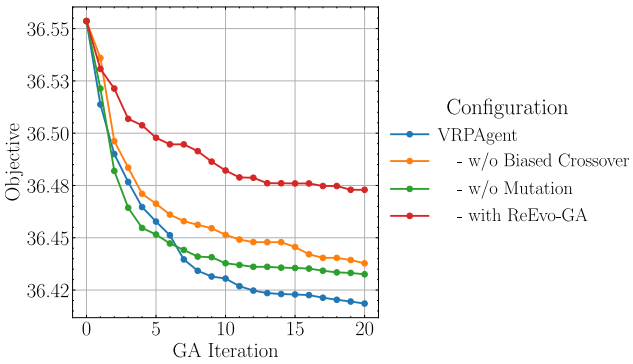
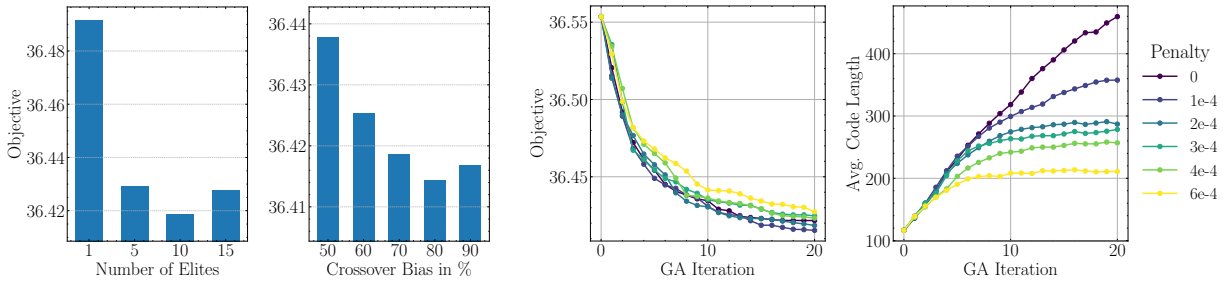



Figure 4: Ablation results.




(a) GA parameters

(b) Code length penalty

Figure 5: Results of the sensitivity analyses.

balances exploitation and exploration and drives faster convergence. Removing mutation lowers final solution quality, and replacing our GA with ReEvo's yields the weakest results, confirming that our combination of elitism, biased crossover, and mutation is essential for discovering high-quality heuristics.

**Crossover Bias and Elite Size** We evaluate the effect of the crossover bias and elite size $M_E$ on the performance of our GA. As shown in Fig. 5a, elite sizes between 5 and 15 yield strong performance, whereas an elite size of 1 leads to drastically worse results by making the search overly greedy. Regarding the crossover bias, a strong preference for the better-performing operator is clearly beneficial, with an operator bias of 80% achieving the best results.

**Code Length Penalty** We investigate the impact of the code length penalty factor $\lambda$ on both the quality and length of the discovered heuristics. Several discovery runs with varying $\lambda$ values reveal that the penalty strongly controls implementation size without substantially degrading performance. For instance, increasing $\lambda$ to $4 \cdot 10^{-4}$ reduces the average length of generated heuristics by roughly 50%, while only causing a marginal drop in objective value (Fig. 5b). These results highlight several key insights. First, the penalty effectively discourages overly long implementations, producing heuristics that are easier for humans to read and maintain. Second, higher penalties reduce token usage during LLM generation, improving efficiency and lowering computational costs. Third, the fact that performance remains largely unaffected even for strong penalties demonstrates that simple, concise heuristics can perform just as well as more complex ones.

**Warm-Starting Heuristic Discovery** We investigate whether operators discovered for one problem can be effectively adapted to other problems by conducting experiments in which we warm-start the discovery process for the VRPTW and PCVRP using the best heuristic found for the CVRP. For the warm-start, we provide the CVRP implementation as an example to the model and task it with adapting the heuristic to the new problem (i.e., VRPTW or PCVRP). Each configuration is run 10 times, and the results are averaged.

Fig. 6 shows the objective function values over the course of the discovery process for the VRPTW and PCVRP, comparing warm-start and cold-start runs. Warm-starting significantly improves performance, yielding better

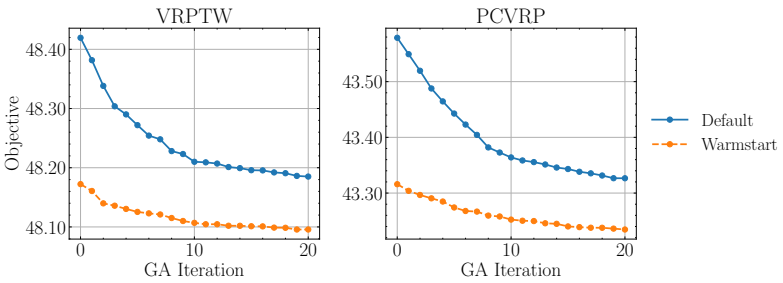



Figure 6: Objective values during heuristic discovery for VRPTW and PCVRP with and without warm-starts from the best CVRP heuristic.

results from the initial population. All warm-start runs begin from the best CVRP implementation found across 10 discovery runs, which required a substantial amount of compute. The combination of high-quality initial heuristics and diversity introduced through the start population generation allows the method to discover implementations that outperform the cold-start runs on average.

## 6 Discovered Heuristic Operators

A key advantage of LLM-driven heuristic discovery over deep learning–based approaches is that the resulting strategies can be directly inspected and audited by human experts. To further facilitate such analysis, we introduce a post-processing stage in which an LLM is used to improve the code quality and readability of the generated implementations. This stage is necessary because code quality is not considered during the discovery process itself: the fitness functions focus solely on performance, and we explicitly instruct the LLMs not to include comments during search. While this reduces interpretability, it substantially lowers token usage and accelerates the discovery procedure.

**Code Quality Improvement** To refine the quality of the discovered heuristics, we pass each implementation to an LLM together with a prompt specifying the desired improvements. The returned code is then evaluated on the validation instances to verify that performance remains unchanged. We repeat this refinement–evaluation loop until we obtain an improved version that matches the original performance. The improvement prompt emphasizes (1) adding comments that explain both high-level heuristic logic and low-level implementation details, (2) restructuring code into clearer functional units, (3) and eliminating unnecessarily convoluted uses of random numbers.

**Analysis** We give the task of analyzing the best discovered heuristic operators for each problem to three OR experts who have years of experience writing heuristics in routing and related fields. The goal of our analysis is to assess the (1) readability, (2) coherence and soundness, (3) maintainability, (4) interpretability, (5) and novelty of the generated heuristic operators. We note that our assessment of the heuristics is not meant to be a thorough scientific analysis that generalizes to other LLMs or optimization problems.

The removal and sorting mechanisms of all three analyzed heuristics can be described as ensemble approaches that use random numbers to choose different (combinations of) heuristics in each iteration. Given the popularity and success of such ensembles in well-known metaheuristics (e.g., adaptive LNS (Pisinger & Ropke, 2018)), it is perhaps not a surprise that we encounter ensembles in the discovered heuristics. Overall, we find the generated heuristics to be relatively easy to interpret and structurally coherent. The comments introduced during post-processing substantially enhance comprehensibility, and the decomposition of the code into functions further improves readability. All of the heuristics generated can be said to be novel. We are not aware of any heuristics in the literature exactly matching these algorithms, however we note that the heuristics do contain some components that are likely inspired by the literature, e.g., SISRs. The detailed analysis including ablation studies for the generated heuristics are available in Appendix E.

## 7 Conclusion

In this work, we introduced VRPAgent, a metaheuristic framework in which LLMs generate VRP-specific operators for a LNS. By constraining the algorithmic discovery to specific operators rather than end-to-end heuristics, VRPAgent makes the task more manageable while achieving strong performance. Using a GA with elitism and biased crossover for algorithm discovery, VRPAgent consistently finds heuristic operators that outperform human-designed approaches on a range of vehicle routing problems. Our results highlight a promising future for automated heuristic discovery in VRPs, suggesting that LLMs could play a key role in designing efficient and adaptable optimization methods for complex, real-world problems.

## References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint*

*arXiv:2508.10925*, 2025. URL https://arxiv.org/abs/2508.10925.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Bk9mxlSFx.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063. URL https://www.sciencedirect.com/science/article/pii/S0377221720306895.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025a. URL https://github.com/ai4co/rl4co.

Federico Berto, Chuanbo Hua, Nayeli Gast Zepeda, André Hottung, Niels Wouda, Leon Lan, Junyoung Park, Kevin Tierney, and Jinkyoo Park. RouteFinder: Towards Foundation Models for Vehicle Routing Problems. *Transactions on Machine Learning Research*, 2025b. ISSN 2835-8856. URL https://openreview.net/forum?id=QzGLoaOPiY.

Edmund K Burke, Matthew R Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pp. 860–869. Springer, 2006.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, Samitha Samaranayake, Carla P. Gomes, and Zhiru Zhang. Heurigym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. *arXiv preprint arXiv:2506.07972*, 2025a. URL https://arxiv.org/abs/2506.07972.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, 2025b.

Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Paulo da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Eren Akçay. Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning. In *Asian Conference on Machine Learning*, 2020.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025. https://github.com/datphamvn/HSEvo.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 77416–77429. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/f445ba15f0f05c26e1d24f908ea78d60-Paper-Conference.pdf.

Shengyu Feng, Weiwei Sun, Shanda Li, Ameet Talwalkar, and Yiming Yang. A comprehensive evaluation of contemporary ML-based solvers for combinatorial optimization. *ArXiv*, abs/2505.16952, 2025. URL https://arxiv.org/abs/2505.16952.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner Takes It All: Training Performant RL Populations for Combinatorial Optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 48485–48509. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/97b983c974551153d20ddfabb62a5203-Paper-Conference.pdf.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 2017.

André Hottung and Kevin Tierney. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. In *European Conference on Artificial Intelligence*, pp. 443–450, 2020.

André Hottung, Mridul Mahajan, and Kevin Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. In *International Conference on Learning Representations*, 2025a.

André Hottung, Paula Wong-Chung, and Kevin Tierney. Neural deconstruction search for vehicle routing problems. *Transactions on Machine Learning Research*, 2025b. ISSN 2835-8856. URL https://openreview.net/forum?id=bCmEP1Ltwq.

Ziwei Huang, Jianan Zhou, Zhiguang Cao, and Yixin Xu. Rethinking light decoder-based solvers for vehicle routing problems. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=4pRwkYpa2u.

Ziyao Huang, Weiwei Wu, Kui Wu, Jianping Wang, and Wei-Bin Lee. Calm: Co-evolution of algorithms and language model for automatic heuristic design, 2025b. URL https://arxiv.org/abs/2505.12285.

Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning Collaborative Policies to Solve NP-hard Routing Problems. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 10418–10430. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/564127c03caab942e503ee6f810f54fd-Paper.pdf.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging Symmetricity for Neural Combinatorial Optimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 1936–1949. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/0cddb777d3441326544e21b67f41bdc8-Paper-Conference.pdf.

Minsu Kim, Sanghyeok Choi, Hyeonah Kim, Jiwoo Son, Jinkyoo Park, and Yoshua Bengio. Ant Colony Sampling with GFlowNets for Combinatorial Optimization. In Yingzhen Li, Stephan Mandt, Shipra Agrawal, and Emtiyaz Khan (eds.), *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, volume 258 of *Proceedings of Machine Learning Research*, pp. 469–477. PMLR, 2025. URL https://proceedings.mlr.press/v258/kim25a.html.

Grigorios D Konstantakopoulos, Sotiris P Gayialis, and Evripidis P Kechagias. Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational research*, 22 (3):2033–2062, 2022.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Xiaozhe Li, Jixuan Chen, Xinyu Fang, Shengyuan Ding, Haodong Duan, Qingwen Liu, and Kai Chen. Opt-bench: Evaluating llm agent on large-scale search spaces optimization problems, 2025a. URL https://arxiv.org/abs/2506.10764.

Yang Li, Jiale Ma, Wenzheng Pan, Runzhong Wang, Haoyu Geng, Nianzu Yang, and Junchi Yan. Unify ml4tsp: Drawing methodological principles for tsp and beyond from streamlined design space of learning and search. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning (ICML)*, 2024a. URL https://arxiv.org/abs/2401.02051.

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. LLM4AD: A platform for algorithm design with large language model. 2024b. URL https://arxiv.org/abs/2412.17287.

Fei Liu, Yilu Liu, Qingfu Zhang, Xialiang Tong, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. *arXiv preprint arXiv:2508.03082*, 2025.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8845–8864. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1c10d0c087c14689628124bbc8fa69f6-Paper-Conference.pdf.

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 11096–11107. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/5c53292c032b6cb8510041c54274e65f-Paper.pdf.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to Search Feasible and Infeasible Regions of Routing Problems with Flexible Neural k-Opt. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 49555–49578. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/9bae70d354793a95fa18751888cea07d-Paper-Conference.pdf.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & operations research*, 51:190–199, 2014.

J Muth. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, 1963.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. URL https://arxiv.org/abs/2506.13131.

NVIDIA Corporation. NVIDIA cuOpt: Gpu-accelerated decision optimization. https://github.com/NVIDIA/cuopt, 2025. Accessed: 2025-09-25.

Wenbin Ouyang, Sirui Li, Yining Ma, and Cathy Wu. Learning to segment for vehicle routing problems. *arXiv preprint arXiv:2507.01037*, 2025.

David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pp. 99–127. Springer, 2018.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Alberto Santini, Michael Schneider, Thibaut Vidal, and Daniele Vigo. Decomposition strategies for vehicle routing heuristics. *INFORMS Journal on Computing*, 35(3):543–559, 2023.

Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2): 139–171, 2000.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, and Jie Zhang. Generalizable Heuristic Generation Through Large Language Models with Meta-Optimization. *arXiv preprint arXiv:2505.20881*, 2025. URL https://arxiv.org/abs/2505.20881.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. Co-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. *ArXiv*, abs/2504.04310, 2025. URL https://arxiv.org/abs/2504.04310.

Anja Šurina, Amin Mansouri, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Scaling Self-Improving Foundation Models without Human Supervision*, 2025. URL https://openreview.net/forum?id=1kAwyBpoO1.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Cong Dao Tran, Quan Nguyen-Tri, Huynh Thi Thanh Binh, and Hoang Thanh-Tung. Large language models powered neural solvers for generalized vehicle routing problems. In *ICLR 2025 Workshop on Towards Agentic AI for Science: Hypothesis Generation, Comprehension, Quantification, and Validation*, 2025. URL https://openreview.net/forum?id=EVqlVjvlt8.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.08.012. URL https://www.sciencedirect.com/science/article/pii/S0377221716306270.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* Neighborhood. *Computers & Operations Research*, 140:105643, 2022.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28, pp. 2692–2700. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024. doi: 10.1287/ijoc.2023.0055. URL https://doi.org/10.1287/ijoc.2023.0055.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning Improvement Heuristics for Solving Routing Problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. HeurAgenix: Leveraging LLMs for Solving Complex Combinatorial Optimization Challenges. *arXiv preprint arXiv:2506.15196*, 2025b. URL https://arxiv.org/abs/2506.15196.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 43706–43728. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/883105b282fe15275991b411e6b200c5-Paper-Conference.pdf.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024a.

Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning Global Partition and Local Construction for Solving Large-Scale Routing Problems in Real-Time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20284–20292, 2024b. doi: 10.1609/aaai.v38i18.30009. URL https://ojs.aaai.org/index.php/AAAI/article/view/30009.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale milp problems. In *Forty-second International Conference on Machine Learning*, 2025.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *International Conference on Machine Learning (ICML)*, 2024a.

Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. UDC: A Unified Neural Divide-and-Conquer Framework for Large-Scale Combinatorial Optimization Problems. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 6081–6125. Curran Associates, Inc., 2024b. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/0b8e4c8468273ee3bafb288229c0acbc-Paper-Conference.pdf`.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=Do1OdZzYHr`.

# A Prompts

The exact prompts used by VRPAgent-GA are presented below. We distinguish between *general prompts*, which remain the same across all problems, and *problem-specific prompts*, which must be tailored to each task. The two are combined by substituting variables in the general prompts (e.g., replacing {problem_desc} with the corresponding problem-specific description). For crossover and ablation prompts, the provided template is further extended by inserting the implementations of the associated individuals at the designated positions.

For brevity, we report only the problem-specific prompts for the CVRP here. The prompts for all problems can be found in our online repository.

## A.1 General prompts

```
You are an operations research expert. Your task is to design new heuristics for an existing **Large Neighborhood
Search (LNS)** framework applied to the {problem_name_long}. The framework iteratively improves a given initial
solution through the following steps:
1. **Customer Removal**: Select a subset of customers to remove using a specified heuristic.
2. **Solution Perturbation**: Remove the selected customers from their tours. This results in an infeasible solution
where the removed customers are no longer served.
3. **Customer Ordering**: Order the removed customers using another heuristic.
4. **Greedy Reinsertion**: Reinsert the removed customers one by one into the tours, following the order defined in
step 3.

Your job is to implement **new heuristics for:**
- **Step 1**: Customer selection (`select_by_llm_1`)
- **Step 3**: Ordering of the removed customers (`sort_by_llm_1`)

All other components of the LNS framework are fixed and **cannot be modified**.

# Routing Problem Description
{problem_desc}

# Other implementation notes and requirements:
- The framework is implemented in **C++**.
- The LNS targets **large instances** (e.g., more than 500 customers).
- Only a small number of customers should be removed in each iteration.
- The selected customers do **not need to form a single compact cluster**, but **each selected customer should be
close to at least one or a few other selected customers**. This encourages meaningful changes during greedy
reinsertion.
- The heuristic must incorporate **stochastic behavior** to ensure sufficient diversity over **millions of
iterations**.
- The search is limited by runtime, meaning that the two new heuristics should be very fast.

# Code style
- IMPORTANT: DO NOT ADD ***ANY*** COMMENTS unless asked
```

Prompt 1: Global system context.

```
[TASK]
Write high-quality heuristics for `select_by_llm_1` and `sort_by_llm_1` in the LNS framework. Write the full code file
in a ```cpp``` code block.

# Example implementation
{seed_code}

# Libary context
You are also provided with some selected header function information with comments that could be useful:
{LNS_headers}
```

Prompt 2: Initial operator generation.

```
[Better Code]
{code_parent_1}

[Worse Code]
{code_parent_2}

[Task]
Write new high-quality heuristics for `select_by_llm_1` and `sort_by_llm_1` in the LNS framework. Your implementation
should be a crossover of the two implementations above, taking most ideas from the better code (80%) and only some
ideas from the worse code (20%).
Ensure that the new code maintains a comparable overall complexity and length to the two implementations above.
Output code only and enclose your code with C++ code block: ```cpp ... ```. Do not comment your code.
```

Prompt 3: Crossover prompt with 80% bias.

```
[Code]
{code}

[Task]
To simplify the heuristics implemented in  `select_by_llm_1` and `sort_by_llm_1` we want to conduct an ablation study.
Choose a random mechanic/component from the code that you think might not be important and remove any trace of it from
the code. We will
then run your code to evaluate the impact of the removed component. Output code only and enclose your code with C++
code block: ```cpp ... ```.
```

Prompt 4: Ablation (Mutation).

```
[Code]
{code}

[Task]
The goal is improve the heuristics implemented in `select_by_llm_1` and `sort_by_llm_1`.
Add a new mechanic/component to the code above. Be innovative. We will
then run your code to evaluate the impact of the new component. Output code only and enclose your code with C++ code
block: ```cpp ... ```.
```

Prompt 5: Extend (Mutation).

```
[Code]
{code}

[Task]
The goal is to find new parameter settings for heuristics implemented in `select_by_llm_1` and `sort_by_llm_1`.
Modify the parameters of the code above to improve the effectiveness of the heuristic. If there are magic numbers in
the code, replace them with constants that are set at the beginning of each function.
Do not make any other changes to the code.
Output code only and enclose your code with C++ code block: ```cpp ... ```.
```

Prompt 6: Adjust-Parameters (Mutation).

```
[Code]
{code}

[Task]
The goal is improve the runtime of the heuristics implemented in `select_by_llm_1` and `sort_by_llm_1`.
Modify the code so that the runtime is reduced. It is ok to slightly change the logic of the heuristic to achieve this.
Output code only and enclose your code with C++ code block: ```cpp ... ```.
```

Prompt 7: Refactor (Mutation).

## A.2 Problem-Specific Prompts

### A.2.1 CVRP

```
The Capacitated Vehicle Routing Problem (CVRP) involves determining a set of delivery routes from a depot to a group
of customers, where each customer has a specific demand and each vehicle has a fixed capacity. The objective is to
design routes that minimize the total distance traveled, while ensuring that:
Each route starts and ends at the depot.
Each customer is visited exactly once by a single vehicle.
The total demand on any route does not exceed the vehicle capacity.

There is no limit on the number of vehicles that can be used.
```

Prompt 8: Problem description CVRP).

```
From `Instance.h`:

```cpp
struct Instance {
    int numNodes; // Total number of nodes including depot
    int numCustomers; // Total number of customers (excluding depot)
    int vehicleCapacity; // Capacity of the vehicle (identical for all vehicles)
    std::vector<int> demand;  // Demand of each node (with the depot at index 0 having a demand of 0)
    std::vector<std::vector<float>> distanceMatrix; //Distance matrix between nodes
    std::vector<std::vector<float>> nodePositions; // Node positions in 2D space
    std::vector<std::vector<int>> adj; // Adjacency list for each node, sorted by distance
}
```

From `Solution.h`:

```cpp
struct Solution {
    const Instance& instance; // Reference to the instance to avoid copying
    float totalCosts; // Total cost of the solution
    std::vector<Tour> tours; // List of tours in the solution
    std::vector<int> customerToTourMap; // Map from each customer to its tour index. This can be used to
    // quickly find which tour a customer belongs to, e.g. solution.tours[solution.customerToTourMap[c]] returns the
      tour of customer c.
}
```

From `Tour.h`:

```cpp
struct Tour {
    std::vector<int> customers; // Customers in the tour, excluding depot
    int demand = 0; // Total demand of the tour
    float costs = 0;  // Total cost of the tour including distance to and from the depot
}
```

From `Utils.h`:
```cpp
int getRandomNumber(int min, int max);
float getRandomFraction(float min = 0.0, float max = 1.0);
float getRandomFractionFast(); // Function to generate a random fraction (float) in the range [0, 1] using a fast method
std::vector<int> argsort(const std::vector<float>& values); // Function to perform argsort on a vector of float values
```
```

Prompt 9: Metaheuristic context ({LNS_headers}).

```
#include "AgentDesigned.h"
#include <random>
#include <unordered_set>
#include "Utils.h"

// Customer selection
std::vector<int> select_by_llm_1(const Solution& sol) {
    // random selection of customers
        std::unordered_set<int> selectedCustomers;

        int numCustomersToRemove = getRandomNumber(10, 20);

        while (selectedCustomers.size() < numCustomersToRemove) {
            int randomCustomer = getRandomNumber(1, sol.instance.numCustomers);
            selectedCustomers.insert(randomCustomer);
        }

        return std::vector<int>(selectedCustomers.begin(), selectedCustomers.end());
}


// Function selecting the order in which to remove the customers
void sort_by_llm_1(std::vector<int>& customers, const Instance& instance) {
    // Placeholder for LLM-based sorting logic
    // This function should implement the logic to sort customers based on a learned model
    // For now, we will just sort randomly as a placeholder
    // sort_randomly(customers, instance);
    static thread_local std::mt19937 gen(std::random_device{}());
    std::shuffle(customers.begin(), customers.end(), gen);
}
```

Prompt 10: Seed heuristic ({seed_code}).

# B Additional Details

## B.1 Costs per Run

Table 3: Comparison token usage and cost estimates across models per run (as of Sep. 2025) with inference providers sources.

| Model | Open Source | Token Usage | | Costs ($) | | Total Costs ($) | Source |
|---|---|---|---|---|---|---|---|
| | | Input | Output | Input | Output | | |
| Gemini 2.0 Flash | ✗ | 2.5M | 1.2M | 0.10 | 0.40 | 0.73 | Vertex AI |
| Gemini 2.5 Flash | ✗ | 4.1M | 7.1M | 0.30 | 2.50 | 18.98 | Vertex AI |
| gpt-oss (120B) | ✓ | 4.0M | 2.5M | 0.09 | 0.36 | 1.26 | Clarifai |
| gemma 3 (27B) | ✓ | 2.5M | 1.2M | 0.09 | 0.16 | 0.42 | DeepInfra |
| Qwen3 (30B) | ✓ | 1.5M | 4.4M | 0.08 | 0.29 | 1.40 | Clarifai |
| Llama 3.3 (70B) | ✓ | 2.3M | 1.0M | 0.08 | 0.29 | 0.47 | Clarifai |

## B.2 Use of Large Language Models

LLMs played an active role in this work. Beyond serving as general-purpose writing assistants for improving clarity, style, and grammar and as coding assistants, LLMs were employed as heuristic discovery tools during the optimization phase of our study. Importantly, the core research contributions, including the design of the framework, theoretical development, and validation of results, were conceived, implemented, and verified exclusively by the authors. All outputs from LLMs were critically assessed, refined, and integrated to ensure correctness and adherence to academic standards.

# C Evaluation on CVRPLib Instances

We evaluate VRPAgent on the X instance set (Uchoa et al., 2017) from CVRPLib, which consists of 100 CVRP real-world-like instances with sizes ranging from 100 to 1000 customers. As a baseline, we report results for the state-of-the-art handcrafted methods SISRs. For both methods, the search time is limited to $1.0 \times n$ per instance, where $n$ denotes the number of customers. For each instance, we perform 10 independent

runs and report the average results. Table 4 summarizes the performance of both approaches on all individual instances.

Table 4: Comparisons on the X set of CVRPLib: best-known solutions (BKS), SISRs and VRPAgent costs, and their percentage gaps. Results show the average performance across 10 runs.

| | | SISRs | | VRPAgent | | | | SISRs | | VRPAgent | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance | BKS | Obj. | Gap% | Obj. | Gap% | Instance | BKS | Obj. | Gap% | Obj. | Gap% |
| X-n101-k25 | 27591 | 27591 | 0.000 | 27591 | 0.000 | X-n336-k84 | 139111 | 139515 | 0.291 | **139377** | **0.192** |
| X-n106-k14 | 26362 | 26383 | 0.080 | **26366** | **0.017** | X-n344-k43 | 42050 | **42110** | **0.145** | 42146 | 0.229 |
| X-n110-k13 | 14971 | 14971 | 0.000 | 14971 | 0.000 | X-n351-k40 | 25896 | **25982** | **0.336** | 25992 | 0.373 |
| X-n115-k10 | 12747 | 12747 | 0.000 | 12747 | 0.000 | X-n359-k29 | 51505 | 51575 | 0.137 | **51570** | **0.127** |
| X-n120-k6 | 13332 | **13332** | **0.000** | 13332 | 0.005 | X-n367-k17 | 22814 | 22866 | 0.228 | **22829** | **0.067** |
| X-n125-k30 | 55539 | 55606 | 0.122 | **55545** | **0.012** | X-n376-k94 | 147713 | 147814 | 0.069 | **147737** | **0.016** |
| X-n129-k18 | 28940 | **28952** | **0.043** | 28955 | 0.052 | X-n384-k52 | 65928 | 66128 | 0.304 | **66066** | **0.210** |
| X-n134-k13 | 10916 | 10943 | 0.249 | **10942** | **0.238** | X-n393-k38 | 38260 | 38393 | 0.349 | **38355** | **0.250** |
| X-n139-k10 | 13590 | 13599 | 0.073 | **13596** | **0.051** | X-n401-k29 | 66154 | 66258 | 0.158 | **66251** | **0.148** |
| X-n143-k7 | 15700 | 15717 | 0.114 | **15716** | **0.103** | X-n411-k19 | 19712 | 19785 | 0.375 | **19764** | **0.268** |
| X-n148-k46 | 43448 | **43478** | **0.070** | 43492 | 0.101 | X-n420-k130 | 107798 | **107891** | **0.087** | 107897 | 0.092 |
| X-n153-k22 | 21220 | **21227** | **0.033** | 21316 | 0.457 | X-n429-k61 | 65449 | 65609 | 0.245 | **65590** | **0.216** |
| X-n157-k13 | 16876 | 16883 | 0.043 | **16876** | **0.000** | X-n439-k37 | 36391 | 36485 | 0.260 | **36461** | **0.193** |
| X-n162-k11 | 14138 | **14153** | **0.112** | 14153 | 0.112 | X-n449-k29 | 55233 | **55420** | **0.339** | 55443 | 0.381 |
| X-n167-k10 | 20557 | **20557** | **0.002** | 20590 | 0.164 | X-n459-k26 | 24139 | 24247 | 0.451 | **24226** | **0.362** |
| X-n172-k51 | 45607 | 45630 | 0.051 | **45607** | **0.000** | X-n469-k138 | 221824 | 222339 | 0.232 | **222249** | **0.192** |
| X-n176-k26 | 47812 | **47841** | **0.061** | 47889 | 0.162 | X-n480-k70 | 89449 | **89568** | **0.134** | 89603 | 0.173 |
| X-n181-k23 | 25569 | 25580 | 0.043 | **25576** | **0.029** | X-n491-k59 | 66483 | 66708 | 0.338 | **66608** | **0.189** |
| X-n186-k15 | 24145 | **24163** | **0.078** | 24180 | 0.145 | X-n502-k39 | 69226 | 69275 | 0.072 | **69272** | **0.066** |
| X-n190-k8 | 16980 | 16996 | 0.094 | **16990** | **0.059** | X-n513-k21 | 24201 | **24295** | **0.392** | 24298 | 0.403 |
| X-n195-k51 | 44225 | 44308 | 0.188 | **44277** | **0.118** | X-n524-k153 | 154593 | 154905 | 0.202 | **154786** | **0.125** |
| X-n200-k36 | 58578 | 58641 | 0.109 | **58635** | **0.098** | X-n536-k96 | 94846 | 95209 | 0.383 | **95153** | **0.324** |
| X-n204-k19 | 19565 | **19631** | **0.342** | 19675 | 0.567 | X-n548-k50 | 86700 | 86806 | 0.123 | **86783** | **0.096** |
| X-n209-k16 | 30656 | **30670** | **0.048** | 30679 | 0.076 | X-n561-k42 | 42717 | 42878 | 0.379 | **42820** | **0.243** |
| X-n214-k11 | 10856 | 10904 | 0.450 | **10897** | **0.380** | X-n573-k30 | 50673 | 50826 | 0.303 | **50773** | **0.198** |
| X-n219-k73 | 117595 | 117624 | 0.025 | **117612** | **0.015** | X-n586-k159 | 190316 | 190699 | 0.201 | **190598** | **0.148** |
| X-n223-k34 | 40437 | 40551 | 0.283 | **40491** | **0.134** | X-n599-k92 | 108451 | 108706 | 0.236 | **108685** | **0.216** |
| X-n228-k23 | 25742 | **25794** | **0.203** | 25797 | 0.215 | X-n613-k62 | 59535 | 59765 | 0.387 | **59679** | **0.242** |
| X-n233-k16 | 19230 | 19282 | 0.272 | **19281** | **0.268** | X-n627-k43 | 62164 | 62335 | 0.275 | **62320** | **0.251** |
| X-n237-k14 | 27042 | **27103** | **0.227** | 27134 | 0.343 | X-n641-k35 | 63682 | 63877 | 0.307 | **63842** | **0.252** |
| X-n242-k48 | 82751 | 82902 | 0.183 | **82879** | **0.155** | X-n655-k131 | 106780 | 106884 | 0.098 | **106825** | **0.043** |
| X-n247-k50 | 37274 | 37389 | 0.311 | **37342** | **0.184** | X-n670-k130 | 146332 | **146992** | **0.451** | 147491 | 0.792 |
| X-n251-k28 | 38684 | 38808 | 0.323 | **38737** | **0.138** | X-n685-k75 | 68205 | 68401 | 0.288 | **68340** | **0.198** |
| X-n256-k16 | 18839 | 18901 | 0.330 | **18885** | **0.246** | X-n701-k44 | 81923 | 82131 | 0.255 | **82080** | **0.192** |
| X-n261-k13 | 26558 | 26657 | 0.375 | **26650** | **0.350** | X-n716-k35 | 43373 | **43491** | **0.272** | 43493 | 0.279 |
| X-n266-k58 | 75478 | **75650** | **0.229** | 75651 | 0.229 | X-n733-k159 | 136187 | 136462 | 0.203 | **136401** | **0.158** |
| X-n270-k35 | 35291 | **35347** | **0.159** | 35362 | 0.201 | X-n749-k98 | 77269 | 77584 | 0.408 | **77493** | **0.291** |
| X-n275-k28 | 21245 | **21275** | **0.145** | 21285 | 0.188 | X-n766-k71 | 114417 | 114761 | 0.301 | **114688** | **0.237** |
| X-n280-k17 | 33503 | 33634 | 0.391 | **33559** | **0.168** | X-n783-k48 | 72386 | 72645 | 0.358 | **72568** | **0.252** |
| X-n284-k15 | 20215 | 20289 | 0.368 | **20272** | **0.283** | X-n801-k40 | 73305 | 73446 | 0.192 | **73438** | **0.183** |
| X-n289-k60 | 95151 | 95452 | 0.317 | **95421** | **0.284** | X-n819-k171 | 158121 | 158513 | 0.248 | **158408** | **0.182** |
| X-n294-k50 | 47161 | 47280 | 0.254 | **47268** | **0.227** | X-n837-k142 | 193737 | 194047 | 0.160 | **194018** | **0.145** |
| X-n298-k31 | 34231 | 34274 | 0.127 | **34264** | **0.097** | X-n856-k95 | 88965 | 89152 | 0.211 | **89119** | **0.173** |
| X-n303-k21 | 21736 | **21788** | **0.239** | 21819 | 0.382 | X-n876-k59 | 99299 | 99544 | 0.247 | **99494** | **0.197** |
| X-n308-k13 | 25859 | 26192 | 1.289 | **25934** | **0.290** | X-n895-k37 | 53860 | 54138 | 0.517 | **54104** | **0.454** |
| X-n313-k71 | 94043 | 94235 | 0.205 | **94189** | **0.156** | X-n916-k207 | 329179 | **329556** | **0.115** | 329566 | 0.118 |
| X-n317-k53 | 78355 | 78397 | 0.054 | **78364** | **0.012** | X-n936-k151 | 132715 | 133387 | 0.506 | **133201** | **0.366** |
| X-n322-k28 | 29834 | 29933 | 0.333 | **29905** | **0.240** | X-n957-k87 | 85465 | 85640 | 0.206 | **85633** | **0.198** |
| X-n327-k20 | 27532 | 27671 | 0.505 | **27636** | **0.379** | X-n979-k58 | 118976 | 119147 | 0.144 | **119118** | **0.120** |
| X-n331-k15 | 31102 | **31141** | **0.127** | 31164 | 0.199 | X-n1001-k43 | 72355 | 72563 | 0.288 | **72536** | **0.251** |
| | | | | | | Avg. Gap% | * | 0.229 % | | **0.195%** | |

# D Success Rate

During the main training runs, we track the proportion of generated operators that function correctly, i.e., those that both compile successfully and produce solutions for all validation instances. We refer to this proportion as the success rate, and it provides insight into the difficulty of the generation tasks posed to the LLMs.

Fig. 7 reports the success rate over the course of the discovery runs. Across all problem settings, the success rate remains above 85%, indicating that the generation tasks are generally manageable for the LLM used. For the VRPTW, we also observe a gradual increase in success as the discovery process progresses. For the PCVRP, the success rate is consistently around 95%, likely because not all customers must be visited, making feasible solution generation comparatively easy (e.g., even a solution with no tours is technically feasible).

Note that the figure reflects the success rate only for operators generated within the main loop of VRPAgent-GA (i.e., offspring and mutation operators). During the initial population generation, the success rates are lower: 73% for the CVRP, 68% for the VRPTW, and 90% for the PCVRP.

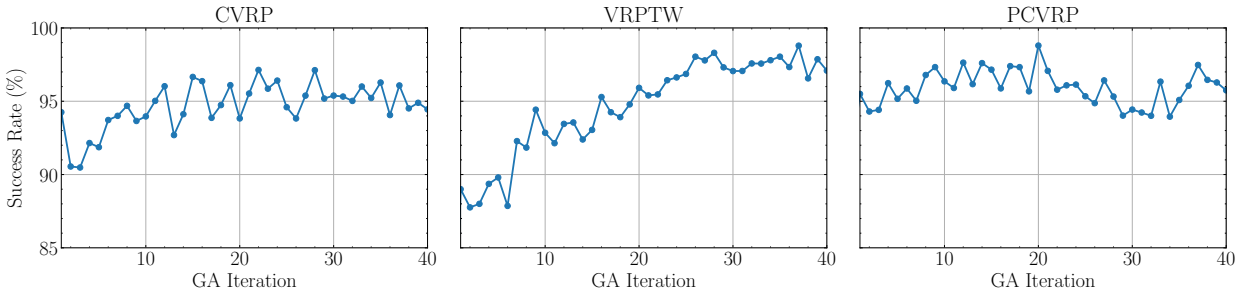



Figure 7: Percentage of operators that compile successfully and produce valid solutions (success rate) throughout the main discovery runs.

# E Analysis of Discovered Heuristics

## E.1 High-level Description

### E.1.1 CVRP

**Customer Removal** The removal heuristic, iteratively constructs a set of customers to remove based on a target size drawn from a uniform distribution (e.g., between 15 and 28). The process initializes by either cutting a random contiguous tour segment (with probability 0.2) or selecting a single random seed. The set is subsequently expanded by choosing a pivot from the currently selected customers and applying one of three weighted strategies: adding a spatial neighbor from the pivot's k-nearest graph (60%), adding a short contiguous segment from the pivot's current tour (25%), or adding the pivot's immediate tour predecessor or successor (15%). A random fallback ensures the target removal count is always met.

**Customer Ordering** The customer ordering operator stochastically selects one of eight sorting rules and a random sorting direction (ascending or descending). The scoring criteria include geometric metrics (distance to depot, distance to the removed set's centroid, polar angle), problem-specific attributes (demand, combined demand-distance), and topological features (connectivity among removed nodes). Alternatively, the order may be determined by a stochastic Nearest-Neighbor chain or a simple random shuffle. To enhance diversity, minor additive noise is applied to the calculated scores during the sorting process.

### E.1.2 VRPTW

**Customer Removal** The removal operator, constructs a cluster of customers to remove by growing a set from an initial random seed up to a target size drawn uniformly between 10 and 15. The expansion logic prioritizes connectivity: at each step, the algorithm attempts to select a tour neighbor (immediate predecessor

or successor in the current route) with a probability of 0.45, or a geographical neighbor from the instance's adjacency list with a probability of 0.35, where the latter is sampled with a power-law bias to favor closer nodes. If these connected expansion attempts fail, or with a 15% probability once the set is sufficiently large, the algorithm performs a "random jump" or falls back to a uniform random selection to ensure the target removal count is met.

**Customer Ordering** The customer ordering operator uniformly selects one of ten available scoring strategies. These strategies range from sorting by single raw attributes (e.g., time window width, start time, or negative demand) to complex composite scores. Specifically, one advanced strategy aggregates five normalized attributes (including service time and distance to depot) using randomly generated weights and polarities, while another assesses demand density relative to time window tightness. To maximize diversity, the final ordering process includes a microscopic noise factor for tie-breaking, a 25% probability of reversing the sorted list, and a final pass that applies a small number of random swaps.

#### E.1.3 PCVRP

**Customer Removal** The removal operator, constructs a cluster of customers to remove by growing a set from an initial random seed up to a target size drawn uniformly between 10 and 20. The seeding process prioritizes unassigned customers (25%) and those currently belonging to tours (60%) before falling back to a uniform selection. The expansion logic is iterative: at each step, a subset of "source" customers is chosen from the current selection, and candidates are gathered from both their topological neighbors (via the adjacency list) and random samples of their current tour-mates. A single candidate is uniformly selected from this pool, with a random safeguard applied if the candidate pool is empty.

**Customer Ordering** The customer ordering operator executes a pure random shuffle with probability 0.1; otherwise, it employs a score-based sort relative to a pivot node. The pivot is selected from the removed set (80%) or set to the depot (20%). The scoring function calculates a weighted sum involving the distance to the pivot, the distance to the depot, the customer's demand, and their prize (if applicable). To enforce diversity, the specific weights for these components are modulated by three distinct parameter regimes—varying the emphasis between prize collection and distance minimization—and the final list is sorted in either ascending or descending order with equal probability.

### E.2 Expert Evaluation

LLM generated code raises many questions about its quality and maintainability. A further question is how the code works and how it is able to achieve state-of-the-art performance. While we are unable to fully answer these questions, we try to provide some initial insights into the quality of the best heuristic generated for each problem. To do this, we asked three experts with many years of experience writing heuristics by hand to analyze the code according to several criteria. We acknowledge that this is not a scientific study and is not intended to draw generalizations about the ability of LLMs to code heuristics for optimization problems. Rather, our goal is to give some indications as to how the code generated compares to code written by humans and what kind of ideas are present.

The three heuristics experts have XX (expert 1), YY (expert 2) and ZZ (expert 3) years of experience writing OR heuristics[2]. All experts have experience with routing problems in addition to other types of OR problems. Each expert provides an evaluation of the generated code of the best performing heuristic for each of the three problems examined in this work. The experts describe a consensus description of how the heuristic works then write independent discussions of each heuristic. The individual rating criteria are as follows:

1. Readability (noting that the assessments are not general statements about LLM code)
2. Coherence and soundness
3. Maintainability
4. Interpretability, i.e., do we know why this code works well?

5. Are there any new ideas in the heuristic?

#### E.2.1 CVRP

The removal and sorting mechanisms are best described as ensembles of heuristics in which the heuristic applied at any given iteration is chosen at random according to a probability distribution determined through the static parameters of the approach. For the selection of customers for removal, the heuristics of the ensemble show a similarity to the SISRs heuristic. In the first, adjacent customers are selected for removal and in the second, random segments of tours are chosen. Since these segments can overlap, we also have a SISRs-like idea. The third heuristic, as best as we can determine, tries to expand a tour segment. For sorting, the heuristic first decides whether to sort descending or ascending according to one of seven different heuristics. We omit a detailed description of all the heuristics, but note that these include generally known ideas for sorting customers in a CVRP, e.g. using criteria such as the distance to the depot, the demand of the customers, weighted combinations of distance and demand, and greedy nearest-neighbor sequencing.

**Expert 1** The code is surprisingly well-written and split into different functions in a logical fashion. The heuristics are coherent and reasonable. The ensemble contains many components, which gives the approach a modular structure. The code appears relatively easy to maintain and follows good C++ design principles. There are no special constructs or libraries used, and the memory management is simple and transparent. While the heuristic largely builds on existing ideas, the scoring mechanism for removed-node connectivity appears novel. The removal heuristic can be seen as a variant of SISRs, but with meaningful differences from the original approach.

**Expert 2** The code is easy to understand and well-written. It contains both meaningful high-level comments that make it easy to grasp the overall structure of the main functions, as well as low-level comments that facilitate understanding of the details. In addition, the code is decomposed into smaller functions with appropriate and descriptive names. Overall, this is good, standard C++ code that should be very maintainable and does not contain any unusual constructs. The selection heuristics can be characterized as an ensemble of various known (and reasonably diverse) approaches that are applied in a random fashion. The sorting function randomly chooses between eight scoring schemes that rely on meaningful criteria; while the individual criteria are mostly standard, some of their combinations for scoring are likely novel.

**Expert 3** The code is generally well-structured, with its components split into separate functions. This organization makes it easier to follow the logic without being overwhelmed by a single large code block. In addition, the descriptive variable names and comments help readers quickly recognize the purpose of each section and provide useful guidance when deeper understanding is needed. Conceptually, the code combines several known ideas from the literature in a structured ensemble; while the individual components are familiar, their combination is somewhat novel and thoughtfully designed.

#### E.2.2 VRPTW

The selection and sorting heuristics form portfolio approaches, that is, ensembles of different heuristics that are probabilistically selected and combined. The selection heuristic removes a randomly chosen number of customers (10-15). It starts with a randomly picked customer and then uses a main loop in which one additional customer is added per iteration, using spatial proximity and tour neighborhood to a selected reference customer as well as pure randomness to guide the choice. The sorting code involves ten different strategies, of which one is randomly chosen per call of the heuristic. Five strategies rely on simple smetrics such as demand or time window start time, four are based on combined metrics and one is a purely random shuffling of the customers. One of the combined metrics is highly complex and heavily parameterized.

**Expert 1** This heuristic is again very easy to read, and the code is nicely structured. The functions are easy to understand, and the code can be considered interpretable. While the sorting criteria largely follow

[2]To avoid potentially violating the double blind submission policy, we do not provide personal information of the experts. These will be provided in the accepted version of this work, and this message will be removed.

established approaches, some combinations of metrics (e.g., the normalized combination in case 7) appear potentially novel and well-motivated.

**Expert 2** As in the CVRP case, the code is very well explained through both high- and low-level comments and is structured into functions in a meaningful way, avoiding deeply nested statements and making it easy to follow and maintain. The code is reasonable and concise modern C++. The selection heuristic randomly applies a set of sensible heuristics, and the sorting code again relies on randomly choosing from a set of useful and well-established sorting criteria, some of which involve weighted combinations of multiple metrics. Although the individual components are relatively simple, the number of heuristics and parameters is noteworthy and reflects a substantial amount of design effort that would be time-consuming to reproduce manually.

**Expert 3** The code is overall readable and, as in the other cases, reflects a consistent and clear coding style. Splitting the implementation into several functions helps users understand the individual components more easily, while the comments and variable names further enhance readability. The sorting section includes many variants, offering flexibility in how different criteria are applied. Overall, the components used are standard for the VRPTW and are combined in a clear and practical manner.

#### E.2.3 PCVRP

The removal and sorting heuristics form an ensemble of heuristics, with the specific method chosen at random according to static parameters defined at the beginning. The removal operator first determines how many customers to remove. It starts the selection from either an unvisited customer, a customer based on a random tour, or a purely random choice. Customers are then added iteratively based on adjacency and tour neighbors, with random selection used as a fallback. The sorting operator either shuffles customers randomly or scores them using a combination of properties such as prize, distance to the depot, and demand, with probabilities applied to vary the weights of these properties.

**Expert 1** The code is well structured, as with the other problems. One noticeable aspect of this implementation is the use of several fixed constants and random values within a specific code block. While the code is already understandable, it could potentially be expressed in a slightly cleaner form. The scoring mechanism itself appears likely to be novel.

**Expert 2** The code is mostly well-written and includes meaningful explanations in both high- and low-level comments. Again, it is reasonably structured into simple and mostly easy-to-understand functions. Both the selection and sorting heuristics seem reasonable and are based largely on known component heuristics and criteria. Compared to the CVRP and VRPTW cases, the LLM-generated sorting function here focuses on two main strategies, one of which uses a weighted combination with weights determined through multiple nested random draws. While this introduces some additional complexity, it appears designed to capture a broad range of behaviors within a compact formulation.

**Expert 3** The code is well-structured and generally readable, though some functions are longer than in the previous heuristics and combine multiple components, similar to what was done in the CVRP and VRPTW implementations. Nevertheless, the code remains easy to follow thanks to the accompanying comments. While the individual components are familiar and standard for this problem domain, some of their combinations are unusual and potentially interesting.

### E.3 Ablation Study

We conduct ablation studies for the best-performing heuristics to better understand their internal mechanisms. To this end, we systematically remove individual subcomponents from the designed heuristics and assess the impact of each change. To ensure statistically meaningful results, we evaluate all variants on larger validation sets comprising 500 instances per problem, using the same 60s per-instance runtime limit as in the main experiments. For every configuration, including the default version (which contains none of the ablated

modifications), we run VRPAgent 10 times on the same set of 500 test instances and record the average tour cost of each run. We report the mean and standard deviation of these run-level averages, together with a 95% confidence interval. Statistical significance relative to the default configuration is determined using the Wilcoxon signed-rank test applied to the paired run averages.

#### E.3.1 CVRP

**Customer Removal** We ablate the following subcomponents that are used by the generated heuristic during the customer removal procedure:

1. **Initial Segment Removal:** This mechanism is employed as an alternative to selecting a single seed customer. It selects a fixed-length contiguous segment of customers for removal to initiate the destruction process. In the subsequent expansion steps, this tour segment is then extended by other customers.

2. **Nearest Neighbor Expansion:** Used in the iterative removal loop, this mechanism grows the removed set by spatial clustering around already-selected customers. It selects a candidate $c$ from the $k$-nearest neighbors of a pivot $p$ using a biased probability distribution, which strongly favors the nearest physical neighbor; this is intended to ensure the final removed set is geographically compact, creating a large, localized "hole".

3. **Adjacent Tour Node Expansion:** This alternative expansion strategy works by identifying the immediate predecessor and successor of a pivot customer $p$ in its current tour and then uniformly selecting one of these adjacent (unselected) nodes. Its intention is to ensure the removal of customers that are tightly coupled in the solution tour.

4. **Route Segment Removal:** This alternative expansion strategy works by identifying a segment of fixed length that includes the pivot $p$ in its tour and adding all nodes in that segment to the removed set.

Table 5 shows the results of the ablation experiments. Removing Route Segment Removal significantly or Nearest Neighbor Expansion leads to a significant increase in costs. Other components, such as Initial Segment Removal and Adjacent Tour Node Expansion, have only minor effects.

Table 5: Ablation of removal strategies for the CVRP.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 36.70378 | 0.003310 | 0.002368 | — |
| - w/o Initial Segment Removal | 36.70551 | 0.003138 | 0.002245 | 0.375000 |
| - w/o Adjacent Tour Node Expansion | 36.70705 | 0.002539 | 0.001816 | 0.064453 |
| - w/o Route Segment Removal | 36.71770 | 0.003249 | 0.002324 | 0.001953 |
| - w/o Nearest Neighbor Expansion | 37.72012 | 0.018678 | 0.013362 | 0.001953 |

**Customer Ordering** We ablate the following subcomponents that are used by the generated heuristic during the customer ordering procedure:

1. **Demand:** A simple ordering rule that orders removed customers based on the customer's demand value in descending order, ensuring high-demand customers are prioritized early during the re-insertion phase.

2. **Center Proximity:** This ordering rule first calculates the geometric center (centroid) of all removed customers, and then sorts them by their Euclidean distance from this centroid.

3. **Depot Proximity:** This ordering rule calculates the score as the raw Euclidean distance from the depot. The list is typically sorted in ascending order of this distance.

4. **Polar Angle:** This rule defines a radial sequencing of insertion priorities. The mechanism calculates the polar angle of each customer relative to the depot (origin) and then sorts the list ascending or descending by this angle to implement a sweep re-insertion strategy.

5. **Connectivity:** This rule prioritizes customers based on their spatial integration with the removed set. The mechanism calculates the score by measuring the connectivity, determined by the sum of inverse distances between each customer and all other customers in the removed set; this is intended to prioritize the re-insertion of customers that are highly integrated with the destroyed cluster.

6. **Weighted Demand-Distance Score:** This rule prioritizes customers based on a weighted sum of demand and distance to the depot. The mechanism calculates a linear combination score, where the weights are varied stochastically and a small noise value is applied to increase exploration.

7. **Nearest Neighbor Chain:** This rule defines a local, greedy sequence. It starts with a random customer and then iteratively finds the **closest unplaced customer** to the most recently selected one until the entire set is ordered; this generates a local, greedy sequence for the destroyed cluster.

8. **Random Shuffle:** This rule simply orders all removed customers at random.

Table 6 shows the results. For most ordering rules, we do not observe a statistically significant impact, which is expected given that many strategies overlap and the effect of each individual rule is limited when numerous strategies are combined. Interestingly, the Connectivity rule stands out as a significant contributor to performance, despite being more computationally expensive than most other strategies.

Table 6: Ablation of ordering strategies for the CVRP.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 36.70378 | 0.003310 | 0.002368 | — |
| - w/o Nearest Neighbor Chain | 36.70083 | 0.001283 | 0.000918 | 0.037109 |
| - w/o Demand | 36.70116 | 0.003045 | 0.002178 | 0.130859 |
| - w/o Weighted Demand-Distance Score | 36.70196 | 0.001920 | 0.001374 | 0.173071 |
| - w/o Random Shuffle | 36.70390 | 0.002159 | 0.001545 | 0.625000 |
| - w/o Polar Angle | 36.70499 | 0.001673 | 0.001197 | 0.431641 |
| - w/o Connectivity | 36.70747 | 0.001244 | 0.000890 | 0.009766 |
| - w/o Center Proximity | 36.70984 | 0.003388 | 0.002424 | 0.003906 |
| - w/o Depot Proximity | 36.75184 | 0.002093 | 0.001498 | 0.001953 |

#### E.3.2 VRPTW

**Customer Removal** We ablate the following subcomponents that are used by the generated heuristic during the customer removal procedure:

1. **Tour Neighbor Expansion:** This mechanism expands the removed set by identifying the immediate predecessor or successor of a pivot customer in its current tour. It is intended to remove chains of customers that are in the same solution tour.

2. **Nearest Neighbor Expansion:** Used to maintain spatial locality, this mechanism selects a candidate from the closest customers of a pivot customer. It utilizes a biased probability distribution that strongly favors the nearest neighbors to ensure the destroyed set remains geographically compact.

3. **Random Jump Expansion:** This mechanism introduces a perturbation to the growth process by selecting a completely random unselected customer, regardless of proximity to the current set. It is only triggered once the removed set reaches a minimum size.

Table 7 presents the results. Removing the Random Jump Expansion has no statistically significant effect on the VRPTW heuristic, while both of the other expansion strategies are critical for maintaining high solution quality.

Table 7: Ablation of selection strategies for the VRPTW.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 47.46744 | 0.005987 | 0.004283 | — |
| - w/o Random Jump Expansion | 47.46464 | 0.004139 | 0.002961 | 0.375000 |
| - w/o Tour Neighbor Expansion | 47.61712 | 0.003621 | 0.002590 | 0.001953 |
| - w/o Nearest Neighbor Expansion | 48.16588 | 0.005831 | 0.004171 | 0.001953 |

**Customer Ordering** We ablate the following subcomponents that are used by the generated heuristic during the customer ordering procedure:

1. **Time Window Width:** This rule orders removed customers based on the width of their time window in ascending order, ensuring that customers with tighter time constraints are prioritized for re-insertion.
2. **Time Window Start:** This rule orders customers strictly by the start time of their time window in ascending order.
3. **Demand:** A rule that orders customers based on their demand value in descending order, prioritizing high-demand customers early in the insertion process.
4. **Depot Distance:** This rule orders customers based on their Euclidean distance from the depot in descending order, prioritizing the re-insertion of outlying customers before those closer to the depot.
5. **Weighted Start and Width:** This mechanism calculates a deterministic score as a linear combination of the time window start time and half the time window width; it prioritizes customers that start early and have tight constraints.
6. **Service Time:** This rule orders customers based on their service duration in descending order, prioritizing those that consume the most route time.
7. **Multi-Attribute Score:** This rule prioritizes customers using a linear combination of five normalized attributes: time window start, time window width, demand, service time, and distance. The weights for each attribute are randomized for every execution to maximize exploration of different sorting criteria.
8. **Arrival Time Slack:** This rule prioritizes customers based on the maximum possible slack between the earliest arrival (determined by distance from depot) and the latest possible arrival (determined by the time window end). It orders customers with larger slack values first.
9. **Weighted Density and Tightness:** This rule calculates a composite score combining demand density (demand divided by service time) and time window tightness (inverse of width). It weights these two components using a stochastic factor to alternate priority between high-density and highly-constrained customers.
10. **Pure Random Shuffle:** This rule simply orders all removed customers at random.

Table 8 shows the results. For most ordering strategies, there is no statistical evidence that they individually have a meaningful impact on the heuristic's performance. This is expected, as 10 different strategies are used and selected uniformly at random during the search, so removing a single strategy has only a minor effect. Notably, the Weighted Start and Width heuristic stands out as an exception, significantly contributing to the overall performance.

#### E.3.3 PCVRP

**Customer Removal** We ablate the following subcomponents that are used by the generated heuristic during the customer removal procedure:

Table 8: Ablation of ordering strategies for the VRPTW.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 47.46744 | 0.005987 | 0.004283 | — |
| - w/o Demand | 47.46509 | 0.003528 | 0.002524 | 0.556641 |
| - w/o Time Window Width | 47.46617 | 0.002892 | 0.002069 | 0.431641 |
| - w/o Arrival Time Slack | 47.46631 | 0.006066 | 0.004339 | 0.695312 |
| - w/o Multi-Attribute Score | 47.46645 | 0.006211 | 0.004443 | 1.000000 |
| - w/o Depot Distance | 47.46671 | 0.007461 | 0.005337 | 1.000000 |
| - w/o Time Window Start | 47.46708 | 0.004177 | 0.002988 | 1.000000 |
| - w/o Weighted Density and Tightness | 47.46756 | 0.005045 | 0.003609 | 1.000000 |
| - w/o Pure Random Shuffle | 47.46765 | 0.006718 | 0.004806 | 0.845703 |
| - w/o Service Time | 47.46789 | 0.007746 | 0.005541 | 0.921875 |
| - w/o Weighted Start and Width | 47.49367 | 0.005458 | 0.003905 | 0.001953 |

1. **Biased Seed Selection:** Instead of selecting the initial seed customer uniformly at random, this mechanism prioritizes the selection of customers that are currently visited by a vehicle over unvisited customers.

2. **Multi-Source Expansion:** Used in the iterative removal process, this strategy selects customers using multiple seed customer, allowing the destruction area to expand in clusters.

3. **Tour-Based Neighbor Expansion:** This mechanism expands the removed set by identifying customers that share the same route as a seed customer. It samples a subset of tour-neighbors to ensure that entire segments of existing routes are targeted are removed.

Table 9 shows the results. All three ablated removal strategies contribute to the overall performance of the heuristic. Interestingly, even subtle algorithmic details, such as the Biased Seed Selection strategy, which starts the customer removal process from previously visited customers with higher probability rather than uniformly at random, can produce small but consistent improvements.

Table 9: Ablation of selection strategies for the PCVRP.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 42.73483 | 0.003547 | 0.002538 | — |
| - w/o Multi-Source Expansion | 42.73821 | 0.001865 | 0.001334 | 0.027344 |
| - w/o Biased Seed Selection | 42.75303 | 0.002260 | 0.001617 | 0.001953 |
| - w/o Tour-Based Neighbor Expansion | 42.97553 | 0.003856 | 0.002758 | 0.001953 |

**Customer Ordering** We ablate the following subcomponents that are used by the generated heuristic during the customer ordering procedure:

1. **Pivot Proximity:** This mechanism orders customers based on their distance to a specific pivot node that is stochastically chosen as either the depot or a random customer from the removed set.

2. **Prize-Focused Scoring:** This configuration corresponds to the first weighting profile. It calculates a weighted sum of the customer's prize and their distance to the depot, assigning significantly higher weight to the prize value. A small noise factor is added to encourage slight exploration, but the primary intention is to prioritize the re-insertion of high-profit customers.

3. **Prize-Distance Scoring:** This configuration corresponds to the second weighting profile. It assigns roughly equal weights to the customer's prize and their distance to the depot. To promote solution diversity a significantly larger noise factor is applied compared to the other profiles.

4. **Distance-Demand Scoring:** This configuration corresponds to the third weighting profile. It shifts the heuristic's focus by heavily weighting the distance to the depot and introducing a weight for customer demand (which is ignored in the other profiles).

5. **Random Shuffle:** This mechanism bypasses the scoring logic entirely and orders all removed customers at random. It is selected with a small probability to increase exploration.

Table 10 shows the results. Interestingly, removing the Prize-Distance Scoring slightly improves performance, making it the only case in our ablation experiments where removal leads to a statistically significant improvement. In contrast, both the Distance-Demand Scoring and Pivot Proximity ordering strategies contribute positively and improve performance in a statistically significant manner.

Table 10: Ablation of ordering strategies PCVRP.

| Configuration | Mean | Std | CI95 | p-value |
|---|---|---|---|---|
| Default Configuration | 42.73483 | 0.003547 | 0.002538 | — |
| - w/o Prize-Distance Scoring | 42.73103 | 0.002071 | 0.001481 | 0.013672 |
| - w/o Prize-Focused Scoring | 42.73353 | 0.002825 | 0.002021 | 0.492188 |
| - w/o Random Shuffle | 42.73397 | 0.001956 | 0.001399 | 0.921875 |
| - w/o Distance-Demand Scoring | 42.74467 | 0.003440 | 0.002461 | 0.003906 |
| - w/o Pivot Proximity | 42.86265 | 0.002567 | 0.001837 | 0.001953 |