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 the first LLM-based paradigm 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 in discovering faster and more 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, such as VRP variants. 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 for general algorithmic design across domains ranging from code synthesis to symbolic planning and mathematical discovery (Madaan et al., 2023). LLMs have proved a 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 of 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, leading 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 an LNS-based metaheuristic in which the problem-specific heuristic operators are generated by an LLM.
- 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 the first LLM-based approach 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 quickly 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 state-of-the-art handcrafted solvers and requires expensive GPU resources, motivating our use of LLM-generated 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 pro-

gramming 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 new 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 a 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 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.

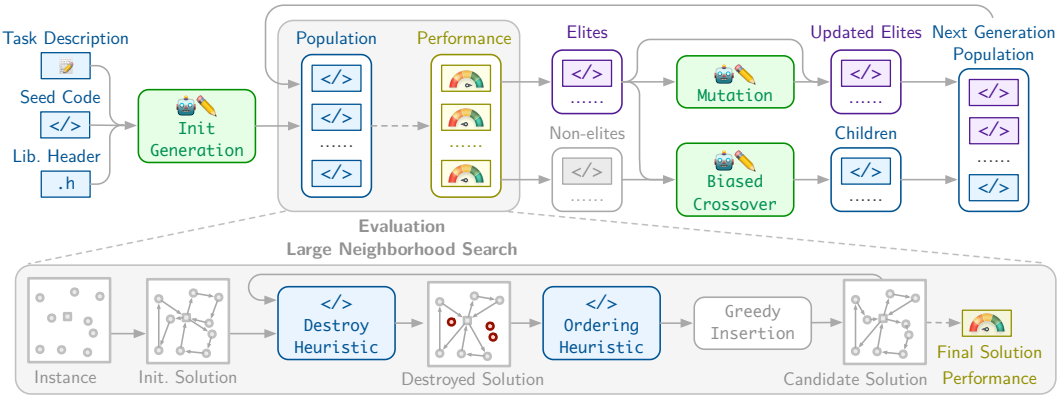


Figure 1: VRPAGENT overview.

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_{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_{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.

Algorithm 1 VRPAGENT-LNS

Input: CVRP Instance l, Destroy Operator f_{REMOVE}, Ordering Operator f_{ORDER}

function LNS(l, f_{REMOVE}, f_{ORDER})
  $s \leftarrow$ GENERATESTARTSOLUTION(l)
  while termination criteria not reached **do**
    $s' \leftarrow f_{\text{REMOVE}}(l, s)$ ▷ Remove some customers from their tours (**LLM-Operator**)
    $insertionOrder \leftarrow f_{\text{ORDER}}(l, s')$ ▷ Order the removed customers (**LLM-Operator**)
    for c in $insertionOrder$ **do** ▷ Reinsert removed customers one by one
      Insert customer c at their locally optimal position in s'
    end for
    $s \leftarrow$ ACCEPT(s, s')
  end while
  return s
end function

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 N_{init}, the number of elites N_{E}, and the number of offspring N_{C} generated in each iteration. The algorithm begins by creating an initial population 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 N_{E} are selected as elites. Next, N_{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 N_{init}, number of elites N_{E}, number of offspring N_{C}

function GA($N_{\text{init}}, N_{\text{E}}, N_{\text{C}}$)
  $P \leftarrow$ GENERATESTARTPOP(N_{init}) ▷ Initialize population
  while termination criteria not reached **do**
    $(E, NE) \leftarrow$ TOP-K-ELITE(P, N_{E}) ▷ Rank heuristics and take the top N_{elite} as elite
    $C \leftarrow \emptyset$
    while $|C| < N_{\text{C}}$ **do**
      $p_e \leftarrow$ RANDOM(E) ▷ Select random elite
      $p_{ne} \leftarrow$ RANDOM(NE) ▷ Select random non-elite
      $c \leftarrow$ BIASED-CROSSOVER(p_e, p_{ne}) ▷ Generate new heuristic via crossover
      $C \leftarrow C \cup \{c\}$
    end while
    for all $e \in E$ **do** ▷ Improve each elite via mutation
      $m \leftarrow$ MUTATION(e) ▷ Modify heuristic using mutation prompt
      if FIT(m) < FIT(e) **then** ▷ Replace elite if improved
        $e \leftarrow m$
      end if
    end for
    $P \leftarrow E \cup C$ ▷ Create next generation
  end while
  return BEST(P)
end function

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^{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 λ 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 interpret and maintain, 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. The best discovered heuristics are made publicly available in our online repository at https://anonymous.4open.science/r/vrpagent-submission.

VRPAGENT Hyperparameters During the discovery phase we use the following parameters unless stated otherwise: elite size $N_E = 10$, offspring size $N_C = 30$, an initial population size of $N_{\text{init}} = 100$. 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.

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), and LKH3 (Helsgaun, 2017). 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 values <1 second and do not benefit from additional search budget. In contrast, ReEvo-ACO combines LLM-generated heuristics with the Ant Colony Optimization (ACO) metaheuristic, and NCO-LLM (Tran et al., 2025) enhances LEHD by automating the design of output logit reshaping and benefit from test-time search.

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 feasible solution were not found for all instances.

	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
	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
	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
	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
	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 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.

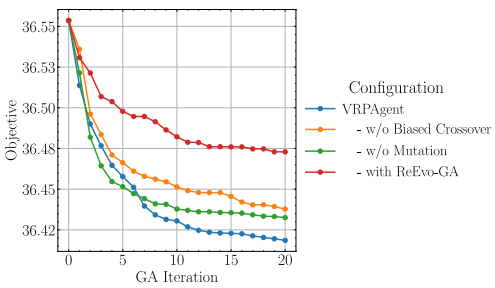


Figure 2: Ablation results.

5.2 ANALYSES

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. 2 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 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.

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. 3 reports the average objective value on the training set during discovery (left) and the total computational cost per run (right). All tested models substantially 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.

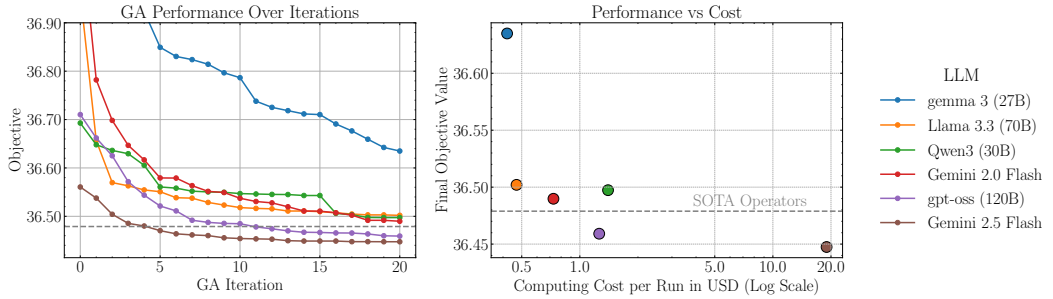


Figure 3: Performance on the CVRP for different LLMs. Detailed cost calculations are provided in Section B.1.

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. 4, VRPAGENT produces heuristics that outperform the state-of-the-art (SOTA) handcrafted operators.

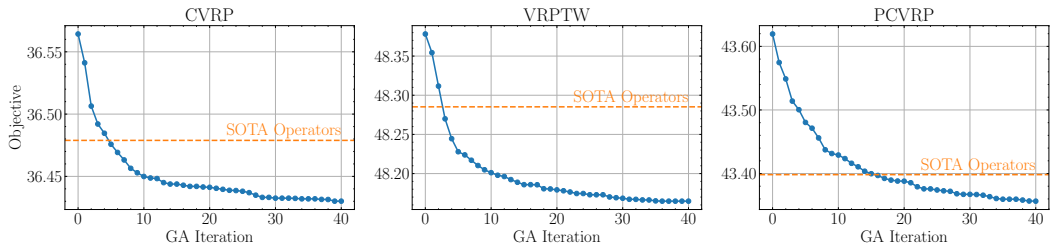
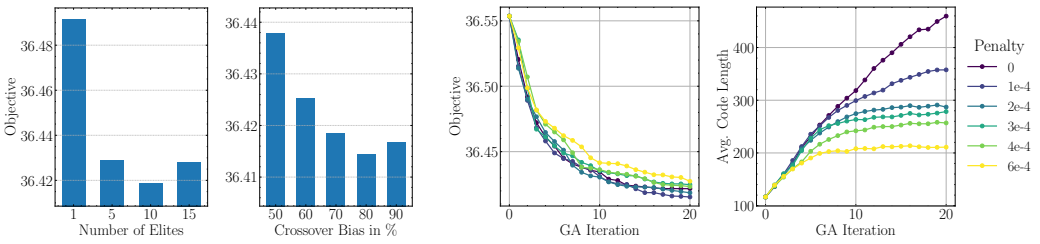


Figure 4: Performance over the course of the discovery process.

Crossover Bias and Elite Size We evaluate the effect of the crossover bias and elite size N_E on the performance of our GA. As shown in Fig. 5a, the best results are achieved with an elite size of 10 and a crossover bias of 80%, indicating that a strong preference for elite mechanics provides a good balance between exploitation and exploration.



(a) GA parameters (b) Code length penalty

Figure 5: Results of the sensitivity analyses.

Code Length Penalty We investigate the impact of the code length penalty factor λ on both the quality and length of the discovered heuristics. Several discovery runs with varying λ values reveal that the penalty strongly controls implementation size without substantially degrading performance. For instance, increasing λ 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 the importance of the penalty: it prevents unnecessarily long, hard-to-interpret implementations, and reduces LLM generation costs, all while preserving high-quality heuristics.

6 Analysis of Discovered Heuristic Operators

We gave the task of analyzing the best discovered heuristic operators for each problem to three coauthors of this paper who have years of experience writing OR 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. We offer a detailed analysis in Appendix C.

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. We assess the generated heuristics as relatively easy to read and very coherent – all three experts read and understood the heuristics without any comments provided by the LLM. The heuristics are maintainable, however the interpretability is difficult given the way the ensembles are coded, especially due to a sometimes convoluted use of random numbers.

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 mainly consist of recombinations of ideas existing in the literature, e.g., SISRs or simple greedy criteria related to distance/demand/time/prizes. Given the complexity of the ensembles, with some having up to nine different component heuristics, a detailed ablation study would be necessary to try to find out which components or combinations of components lead to good performance.

7 Conclusion

In this work, we introduced VRPAgent, a metaheuristic framework in which LLMs generate problem-specific operators for a LNS. By focusing on operator generation rather than end-to-end heuristics, VRPAgent makes the discovery 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, suggesting that LLMs could play a key role in designing efficient and adaptable optimization methods for complex, real-world problems. For future work, we will investigate how to further simplify the generated heuristics to help increase the ease of using VRPAgent generated code in practice.

Reproducibility Statement

We have made every effort to ensure the reproducibility of our results. Detailed descriptions of configurations, prompts, the discovery pipeline, and overall experimental setups are provided in both the main paper and the appendix to enable independent reproducibility. All code to reproduce the experiments will be made open-source upon acceptance.

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.

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.

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.

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, while the rest will be provided in the final code release.

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}).

```cpp
#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 2: 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 Discussion of discovered heuristics

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 have three co-authors of the paper with many years of experience writing heuristics by hand 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. We note that the code is generated without comments to avoid the LLM influencing the analysis, however we note that variable names are present that do give some contextual information about what the code does.

The three heuristics experts have XX (expert 1), YY (expert 2) and ZZ (expert 3) years of experience writing OR heuristics[1]. 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?

C.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.

C.1.1 Expert 1

The code is generally easy to read and understand, although there are many sanity checks that might be unnecessary. The heuristics are rather coherent and are rather reasonable, however note that the removal mechanism likely could be simplified as the heuristics somewhat overlap in what they do. An ablation analysis of the sorting would likely also show that not all of the heuristics are necessary. The code looks relatively easy to maintain as it does not use any special constructs or libraries, and furthermore its memory management is very simple. A detailed ablation analysis would be necessary to find out why the code works well. Finally, the heuristic does not present anything radically new, but there are not any sorting heuristics like this in the literature. The removal heuristic, as mentioned, is a SISRs variation so its novelty is low.

C.1.2 Expert 2

The code is relatively easy to understand and mostly makes sense. In many places, however, it is unnecessarily complex and redundant. A human coder would certainly come up with a better-to-understand and more concise implementation of (almost) the same heuristic. As an example,

[1] To avoid potentially violating the double blind submission policy, we do not indicate the years of experience of the experts, as they are all coauthors of the work. These will be provided in the accepted version of this work, and this message will be removed.

the probabilistic strategy choices involve deeply nested if-else statements and could be managed in a simpler way. Regarding redundancy, the initialization phase and the main strategy loop of the selection heuristic randomly opt for a segment-based selection; I don't think removing it from the initialization would make any difference. The code involves several parameters (including strategy selection parameters) which are mostly interpretable; I cannot assess if the parameters are well-chosen without further experiments. Overall, all the elements of the ensemble are variations of known strategies, but to the best of my knowledge, the chosen combination is new.

C.1.3 EXPERT 3

The code is overall well-structured and, thanks to informative variable names, fairly easy to follow. However, the extensive nesting of loops and conditionals can make it difficult to trace the logic, particularly for less experienced users. If rewritten by an expert, the implementation would likely be shorter and more concise. The large number of parameters and probability values further complicates interpretability, as it is not immediately clear which of them actually matter in practice. Conceptually, the code combines several known ideas from the literature in a structured ensemble; while none of the components are new by themselves, the way they are combined is somewhat novel.

C.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.

C.2.1 EXPERT 1

This heuristic is again fairly easy to read, however a human would likely select the random number differently for determining which heuristic of the ensemble to use for the removal operator. Both the removal and sorting operators are coherent with a clear structure. The removal operator could likely be simplified without losing the core ideas. The methods are, as in the CVRP, not very interpretable and require significant ablation to figure out what actually works. The sorting criteria are mostly standard and there are no big surprises, even though some of the combinations of criteria might be unusual.

C.2.2 EXPERT 2

The code is not hard to read and the variable names are mostly interpretable. There are no substantial errors, and the heuristics are mostly coherent, but the logic is at times unnecessarily hard to understand and may lead to misinterpretations. As an example, there are parameters referring to probabilities of strategies in the selection step which may be easily misinterpreted as they are applied in a nested fashion. In addition, parts of the code regarding random selection are unnecessarily complicated and highly redundant. The sorting code is sometimes hard to read because certain parts belonging to the same strategy are spread across the function body, and declaring certain variables on a global scope does not contribute to maintainability. The sorting criteria are well-known, but two combined criteria are very complex, including weights and stochastic perturbations that make it very hard to assess, maintain and tune.

C.2.3 EXPERT 3

The code is overall readable and, as in the other case, reflects a consistent style the LLM uses. The nested conditionals make the logic harder to follow, and if rewritten manually the implementation would likely be shorter and clearer. The removal procedure is coherent and grounded in known ideas, though the many probabilistic parameters and fallback rules make its behavior less transparent. The

sorting part includes many variants, but the large number of options risks obscuring which criteria truly matter. Overall, the components are standard for the VRPTW.

C.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.

C.3.1 EXPERT 1

The ensembles for removal and sorting are relatively clear, but there are some "magic numbers" strewn about the code. The code is generally coherent, but the removal heuristic gets rather complicated with its choice of random customers. The sorting uses many random numbers and if a student wrote this for me, I would demand a re-write. These heuristics are again mostly combining multiple elements that are already known into a random selection ensemble. It is exceptionally hard to say where the good performance of this heuristic is coming from.

C.3.2 EXPERT 2

The code is generally easy to read and to interpret, but in some cases prone to a misinterpretation of parameters representing probabilities. The code mostly appears coherent and makes reasonable choices, but the implementation is often unnecessarily complex and feels over-parameterized. The elements contributing to the ensemble are mostly straightforward and known in the literature. In the sorting heuristic, tunable parameters are not located in a single location at the beginning of the code, but spread across the code; in some cases, important parameters are not even variables. Interestingly, all three main sorting strategies use the same criteria (all criteria are relatively standard), although with different weight combinations. I cannot really make sense of the weight generation, and it would certainly be hard to tune given the way it is implemented.

C.3.3 EXPERT 3

The code is well-structured and generally readable, though some parts, such as the selection of the first customer, appear overcomplicated and could likely be simplified without losing functionality. The overall structure is consistent, but it relies on many nested components, which can make the logic harder to follow and maintain. Sorting uses numerous hard-coded values, which make the impact of the introduced probabilities not clear. While the components themselves are familiar and standard for this problem domain, some of their combinations are unusual.