Methodological Description for CVRPLIB Challenge Registration

Team Name: UnixCont VRP **Affiliations:** UNICAL **Team Members:** F.P.S.

Date: January 2026

1. Abstract

This document outlines the methodology developed by our team for the CVRPLIB BKS Challenge. Our approach, tentatively titled **"Hybrid GPU-Accelerated Framework with Reinforcement Learning (H-GRL)"**, addresses the Capacitated Vehicle Routing Problem (CVRP) by integrating a novel constructive heuristic based on the Knapsack Problem with a massively parallel Local Search engine running on GPU. Furthermore, an adaptive Reinforcement Learning (RL) agent governs the selection of local search operators to escape local optima efficiently. The method prioritizes capacity saturation in the early stages and distance minimization in the refinement stages.

2. Introduction

The CVRP remains one of the most challenging combinatorial optimization problems. Modern state-of-the-art solvers often rely on Hybrid Genetic Search (HGS) or Branch-and-Cut-and-Price algorithms. Our proposed method seeks to diverge from the standard "Cluster-First, Route-Second" or standard HGS paradigms by introducing a **"Saturation-First, Repair-Later"** constructive phase, followed by a high-throughput improvement phase leveraging hardware acceleration.

Our framework consists of three distinct stages:

1. **Constructive Phase:** A Knapsack-based insertion heuristic with a global "Steal & Restart" repair mechanism.
2. **Intensification Phase:** A GPU-accelerated Local Search performing massive neighborhood evaluation.
3. **Diversification Phase:** A Reinforcement Learning agent for Adaptive Operator Selection (AOS).

3. Constructive Phase: Knapsack-Based Heuristic with Cascade Restart

To generate high-quality initial solutions, we developed a deterministic heuristic that focuses on minimizing the number of vehicles used (fleet minimization) before optimizing for distance.

3.1 Knapsack Filling Strategy

Instead of standard sequential insertion (e.g., Nearest Neighbor), we model the filling of each vehicle as a **Knapsack Problem**.

- **Sorting:** Unserved customers are sorted by demand in descending order.

- **Selection:** For the current vehicle k, we solve a Knapsack sub-problem to identify the subset of unserved customers that maximizes the vehicle's load utilization ($\sum d_i \le Q$). This ensures that vehicles are typically saturated to capacity limits.
- **Routing:** The selected subset is ordered using a 2-opt heuristic to form a valid route.

3.2 Proximity-Based "Steal" and Global Restart

A pure Knapsack approach often neglects geographical proximity. To mitigate this, we implemented a **"Steal & Refill"** mechanism:

1. **Incremental Insertion:** Customers identified by the Knapsack solution are inserted one by one.
2. **Proximity Check:** After inserting a node N into the current vehicle V_{curr}, the algorithm checks all previous vehicles V_{prev} for nodes M that are geographically close to N.
3. **The Steal:** If moving M from V_{prev} to V_{curr} results in a global distance reduction (First Improvement policy), the move is executed.
4. **Cascade Restart:** Crucially, if a "Steal" occurs, the algorithm creates a "hole" in a previous vehicle. To prevent inefficiency, the algorithm halts processing V_{curr} and **restarts the entire construction process** starting from the refinement of the now-modified V_{prev}. This ensures that earlier routes remain tightly packed and optimized before new routes are finalized.

4. Intensification Phase: GPU-Accelerated Local Search

Once an initial pool of solutions is generated, we employ a **CUDA-based parallel architecture** to perform Local Search. Given the computational density of neighborhood evaluation, GPUs offer a significant throughput advantage over CPU-based sequential evaluation.

4.1 Parallel Neighborhood Exploration

The solution is transferred to GPU memory. We implement a **SIMT (Single Instruction, Multiple Threads)** approach where:

- Each thread block manages a specific route or a pair of routes.
- Millions of moves are evaluated in parallel.
- Supported operators include: **2-opt**, **Relocate**, **Swap**, **Cross-Exchange**, and **2-opt*** (inter-route).

4.2 Massive Batch Evaluation

Unlike traditional LS which evaluates neighbors sequentially, our GPU kernel computes the delta costs of all possible moves in the $N \times N$ neighborhood matrix simultaneously.

The move with the best improvement is selected and applied (Best Improvement strategy), updating the distance matrix and route pointers in VRAM.

5. Diversification Phase: Reinforcement Learning (RL)

To avoid stagnation in local optima, we integrate a **Deep Reinforcement Learning (DRL)** agent trained to perform Adaptive Operator Selection (AOS).

5.1 Learning Architecture

- **Model:** We utilize a Proximal Policy Optimization (PPO) agent.
- **State Representation:** The state is defined by feature vectors representing the current solution quality, including average route fill rate, spatial density of routes, and stagnation counter (number of iterations without improvement).
- **Action Space:** The agent selects which perturbation heuristic (ruin-and-recreate, large-scale swap, or specific crossover operators) to apply next.
- **Reward Signal:** The reward is proportional to the improvement in the objective function (total distance) over a fixed time window.

This learning component allows the solver to "learn" the topology of the specific instance class (e.g., clustered vs. random distribution) during the search process, dynamically adjusting the search trajectory.

6. Implementation and Acknowledgments

6.1 Technology Stack

- **Core Logic:** C++ for performance-critical components.
- **Heuristic Prototyping:** Python (as demonstrated in preliminary testing).
- **Optimization Solvers:** Gurobi Optimizer is used for solving the exact Knapsack sub-problems during the constructive phase.
- **GPU Computing:** CUDA C++ / CuPy for the parallel local search kernels.
- **Machine Learning:** PyTorch for the RL agent implementation.

6.2 Acknowledgments and Third-Party Code

In compliance with the competition rules, we acknowledge the use of concepts and open-source libraries.

- The **Knapsack-based construction** logic is original to our team.
- The **GPU parallelization** strategy borrows concepts from *Cappart et al.* on neural combinatorial optimization.

- We utilize **Gurobi** (under academic/commercial license) for exact sub-problem resolution.
- Structure for solution management may draw inspiration from the open-source **HGS-CVRP** (Vidal et al.) repository, adapted significantly to support the "Steal & Restart" logic and GPU offloading.

7. Conclusion

Our method represents a fusion of classical operations research techniques (Knapsack formulations) with modern high-performance computing (GPU parallelism) and artificial intelligence (RL). We believe the unique "Restart from Zero" logic in the constructive phase provides a superior starting point for the solver, allowing the GPU-accelerated refinement to converge rapidly to near-optimal solutions.