# IMITATION IMPROVEMENT LEARNING FOR LARGE-SCALE CAPACITATED VEHICLE ROUTING PROBLEMS

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

## ABSTRACT

Recent works using deep reinforcement learning (RL) to solve routing problems such as the capacitated vehicle routing problem (CVRP) have focused on improvement learning-based methods, which involve improving a given solution until it becomes near-optimal. Although adequate solutions can be achieved for small problem instances, their efficiency degrades for large-scale ones. In this work, we propose a new improvement learning-based framework based on *imitation learning* where classical heuristics serve as experts to encourage the policy model to mimic and produce similar or better solutions. Moreover, to improve scalability, we propose *Clockwise Clustering*, a novel augmented framework for decomposing large-scale CVRP into subproblems by clustering sequentially nodes in clockwise order, and then learning to solve them simultaneously. Our approaches enhance state-of-the-art CVRP solvers while attaining competitive solution quality on several well-known datasets, including real-world instances with sizes up to 30,000 nodes. Our best methods are able to achieve new *state-of-the-art* results for several large instances and generalize to a wide range of CVRP variants and solvers. We also contribute new datasets and results to test the generalizability of our deep RL algorithms.

## 1 INTRODUCTION

We study the vehicle routing problems (VRP), an important class of combinatorial optimization problems which has a wide range of applications in logistics (Laporte, 2009). Capacitated vehicle routing problem (CVRP) is a basic variant of VRP, aiming to find a set of routes that minimize the cost and fulfill the demands of a set of customers without violating vehicle capacity constraints. The CVRP is NP-hard (Dantzig & Ramser, 1959; Lenstra & Kan, 1981), and both exact and heuristic methods have been developed to solve it (Golden et al., 2008; Kumar & Panneerselvam, 2012; Toth & Vigo, 2014). In recent years, especially after the seminal work of Pointer Networks (Vinyals et al., 2015) and Graph Neural Networks (Prates et al., 2019), researchers have started to develop new deep learning and reinforcement learning (RL) frameworks to solve combinatorial optimization problems (Nazari et al., 2018; Bello et al., 2017; Khalil et al., 2017; Velickovic et al., 2018; Kool et al., 2019; Chen & Tian, 2019). The idea behind the RL algorithms is that a machine learning method could learn better heuristics by extracting useful information directly from data, rather than having an explicitly programmed behavior like heuristic methods. In fact, it has been known that, even though heuristics would get stuck in local optimums, they often offer stable and high-quality solutions, especially for large-scale instances and when the problem involves complex constraints. The RL approach is better in exploring new solutions and escaping from local optimums, but needs time to train and would be unstable with complex constraints. This motivates us to combine heuristics and RL in such a way that RL can learn and benefit from heuristic operators.

In this work, we propose an imitation reinforcement learning algorithm trained via policy gradient to learn improvement heuristics based on $k$-opt moves and treat advanced heuristics (e.g., VNS or HGS) as experts to *teach* the policy model. Our work aims to enhance the deep RL process via heuristic methods and address scalability by learning from smaller sub-problems simultaneously. That is, for each step, a certain amount of nodes are selected in turn clockwise, following by an initial solution, to form sub-problems to our RL policy for ease of learning. In fact, our *clockwise* mechanism offers a good initial solution structure and a natural way to decompose the whole problem into sub-problems that can be solved and learned simultaneously. Moreover, a solution returned

by the RL policy model will be fed to a heuristic method to be further improved. Solutions from heuristics are also collected to build an imitation learning model that will be integrated back into the RL policy to persuade the RL to produce similar solutions.

In summary, by combining the clockwise framework, heuristic methods and imitation learning, we bring several advantages to the same place:

(i) Starting with poor quality solutions, we first use heuristics to improve solutions, and heuristic operators are learned and imitated by deep learning networks (i.e., *generative adversarial imitation learning* (Ho & Ermon, 2016)) to generate similar or better solutions at each step. In fact, by letting deep RL and imitation learning work together in an iterative manner, we encourage the algorithm to both explore new solutions (by deep RL) and exploit operators that lead to high-quality solutions (i.e. imitation learning mimicking heuristics).

(ii) The whole network is clustered into smaller sub-problems of similar distributions, allowing our algorithms to process them quickly and scale up.

(iii) Heuristics play as experienced experts (or teacher/corrector) that help the deep RL generate more stable and high-quality solutions. Faster convergence is to be expected with the powerful performance of heuristic methods. Moreover, since solutions are always corrected by heuristics to ensure feasibility, the Policy Network doesn't need to control infeasible solutions.

Altogether, we make the following contributions:

- We propose *Imitation Improvement Learning*, a new learning-based framework using policy gradient with a heuristic method that serves as an expert/teacher to correct and improve any solution to ensure feasibility and teach the policy gradient to generate high-quality solutions.
- We propose *Clockwise Clustering*, a recursive sub-problem decomposition framework to handle large-scale CVRP instances.
- We offer new state-of-the-art solutions for some well-known large-scale CVRP instances.

Our experiments demonstrate the scalability of our *Clockwise Clustering* in solving large-scale real-world instances, the benefits of using imitation learning to get high-quality solutions, and the generalizability of our learning-based model in solving instances of similar distributions.

## 2 Related Work

Heuristics for solving combinatorial optimization problems have been developed for decades. The most powerful methods, such as local search (Crama et al., 1995), genetic algorithms (Heiss-Czedik, 1997), and ant colony methods (López-Ibáñez, 2010), involve iteratively improving solutions in a hand-designed neighborhood search. For example, *move*, *swap* (Wu et al., 2016), and *2-opt* (Croes, 1958) are well-known heuristics for the traveling salesman and vehicle routing problems. Examples of the state-of-the-art heuristic algorithms for the CVPR would be the *HGS* (Vidal, 2022; Vidal et al., 2012) that uses a hybrid genetic and local search procedure to achieve state-of-the-art solution qualities on instances of sizes up to 1000, or the *LKH-3* (Helsgaun, 2017) that uses the Lin-Kernighan heuristic (Lin & Kernighan, 1973) as a backbone, which involves swapping pairs of sub-routes to create new routes. For large-scale problem instances, low-level heuristics are often combined with meta-heuristics to achieve good performance, e.g., Tabu Search with Adaptive Memory (Taillard et al., 2001), Knowledge-Guided Local Search (Voudouris & Tsang, 2003), Large Neighborhood Search (Shaw, 1998), Quantum Annealing (Syrichas & Crispin, 2017), Pruning and Sequential Search (Arnold et al., 2019), Spatial Partitioning Strategies (Tu et al., 2017), Constrained Clustering (Alesiani et al., 2022) and Cluster-First Route-Second (Shalaby et al., 2021).

In recent years, there have been a number of studies focusing on using deep RL to solve combinatorial optimization problems. Those models are categorized into two classes, i.e., *construction* and *improvement methods* (Kwon et al., 2020):

- *Construction methods* (Nazari et al., 2018; Kool et al., 2019): Starting with an empty solution, a construction method constructs a solution by sequentially assigning each customer

to a vehicle until all customers are served. Construction methods still require additional procedures such as beam search, classical improvement heuristics, and sampling to achieve such results.

- *Improvement methods* (Chen & Tian, 2019; Hottung & Tierney, 2019): Starting with a complete initial solution, the methods select either candidate *nodes* (customers or depot) or heuristic *operators* (or both) to improve and update the solution at each step. This is repeated until termination. Here, if one can learn a policy to improve a solution, such a policy can be used to obtain better solutions from a construction heuristic or even random solutions. Studies have shown that *improvement* methods are able to provide better solutions than *construction* ones (Lu et al., 2020; da Costa et al., 2021).

Deep RL approaches have achieved competitive results as compared to classical heuristics. For example, Lu et al. (2020) propose *Learning-to-improve* based on Meta-controller learning, which outperforms LKH-3 but only works on small-scale problems. da Costa et al. (2021) propose *Learning 2-opt* based on learning from local search operations, which also only works with small-scale problem instances. Recently, Li et al. (2021) develop a *learning to delegate* approach in which subproblem are selected and learned. This method outperforms LKH-3 and works well with uniformly large-scale problems. It is moreover quite scalable and is efficient for generalization. However, since the approach requires a large dataset of instances for training and similar data distributions for testing, the performance on non-uniform large-scale CVRP instances such as CVRPLIB would be poor. In fact, there are very few learning-based experimental studies on very large-scale instances. Although deep RL's learnability is appealing, trajectory collection becomes prohibitively expensive for large-scale problem instances.

# 3 BACKGROUND

## 3.1 CAPACITATED VEHICLE ROUTING PROBLEMS

CVRP can be defined by a fully connected weighted graph $\mathcal{G} = (\mathbb{V}, \mathbb{A})$, where $\mathbb{V} = \{0 \cup \mathbb{I}\}$ stands for a set of nodes and $\mathbb{A} = \{(i, j)|i, j \in \mathbb{V}, i \neq j\}$ denotes a set of arcs connecting these nodes. Set $\mathbb{I}$ denotes the set of customers and 0 denotes the central depot. Each arc of the network is associated with a non-negative value $d_{ij} = \|x_i - x_j\|_2$ representing the distance between two nodes $i$ and $j$, where $x_i$, $x_j$ are vectors of spatial coordinates of nodes $i$ and $j$, respectively. Each customer $k \in \mathbb{I}$ is assigned a positive demand $b_k > 0$. At central depot, the demand $b_0$ is set to 0. We also let $\mathbb{B} = \{b_k|k \in \mathbb{I}\}$ denote the set of demands. The objective function of the CVRP, assuming that the fleet of vehicles is homogeneous, is to seek a set of routes that minimize the total traveling distance such that each customer is visited exactly once by exactly one vehicle. Vehicles start and end at the depot and for every route, the total demand of customers does not exceed the maximal carrying capacity $C > 0$ of the vehicle. In Appendix A.1 we provide a more detailed description and a mixed-integer formulation for the CVRP problem.

## 3.2 $k$-OPT HEURISTIC FOR THE CVRP

An improvement heuristic concerns a search procedure that iteratively improve feasible solutions. A procedure can start with an initial solution $S_0$ and iteratively search a better solution $S_{t+1}$ from a current solution $S_t$. Local search methods such as Lin-Kernighan-Helsgaun (LKH) (Helsgaun, 2017) performs well for CVRP. The procedure seeks for $k$ edge swaps ($k$-opt moves) that could be replaced by new edges to form a shorter tour. Sequential pairwise operators such as $k$-opt moves can be decomposed in simpler $k'$-opt ones ($k' < k$). For instance, sequential 3-opt operations can be decomposed into one, two or three 2-opt operations. However, in local search algorithms, the quality of the initial solution usually affects the quality of the final solution, i.e. local search methods can easily get stuck in local optima (Hansen & Mladenović, 2006). To avoid local optima, different meta-heuristics have been proposed, including Simulated Annealing and Tabu Search, which work by accepting bad solutions to enhance exploration on the searching space. Meta-heuristics, nevertheless, still require expert knowledge and rely on sub-optimal rules in their designs.

## 4 CLOCKWISE CLUSTERING

In this section we formally introduce the "Clockwise Clustering" framework, our method for solving large-scale CVRPs. Figure 5 illustrates an overview of this framework. From a large-scale instance input $(\mathcal{G}, \mathbb{B}, C)$, we generate an initial solution $S_0$ by a *clock-hand initializer*, an simple procedure that arranges all nodes in clockwise order and sequentially groups nodes into tours satisfying that the total demand of each tour is not greater than the demand capacity $C$. Clock-hand initializer basically produces a feasible solution that all tours $T = [t_1, t_2, ..., t_{n_T}]$ are ordered in clockwise direction. Note that sorting tours in clockwise direction means that tours are arranged by the angle between y-axis and the line connecting the centroid point and the depot of each tour. The first $u$ tours (i.e., a proportion of all the tours) $[t_1, t_2, ..., t_u]$ are then selected to form a sub-instance $(\mathcal{G}_{sub}, \mathbb{B}, C)$. Here, only a subset of tours is selected to process, instead of all the tours, to enhance the scalability. After solving this sub-instance by the *Imitation Improvement Learning* framework (described in the next section), we then send the first $v$ tours of the returned sub-solution to *a sub-solution buffer* to keep the results. The unprocessed tours and nodes are then collected and sent back to the beginning of the cycle loop for the next round of the *Imitation Improvement Learning*. When all the nodes are processed, we take the processed tours from the *sub-solution buffer* to build a complete (and feasible) solution $S_1$ of $\mathcal{G}$. After that, $S_1$ could be processed same as $S_0$ to create new solutions $S_2$, $S_3, S_4, ....$ We accept bad solutions to allow more exploration on the search space. After a certain number of loops, we return the best solution among $\{S_1, S_2, S_3, ...\}$. We provide an overview of the framework in Appendix A.2.

The *clock-hand initializer* offers a good initial solution and a natural way to decompose the whole problem into sub-problems of similar distributions, allowing heuristics to process them quickly and providing RL with sub-instances of similar distributions for efficient training. The *learning-to-delegate* framework (Li et al., 2021) also endeavors to select sub-problems to improve scalability, but differ from our approach by the fact that the *learning-to-delegate* learns to select sub-problems and uses traditional heuristics (HGS/LKH3) with a huge number of running steps to achieve good sub-instance selections. Instead, our method does not requires such a large number of steps as the sub-instance decomposition is embedded as part of the heuristic loop and cooperates with heuristics so both will be improved over iterations.

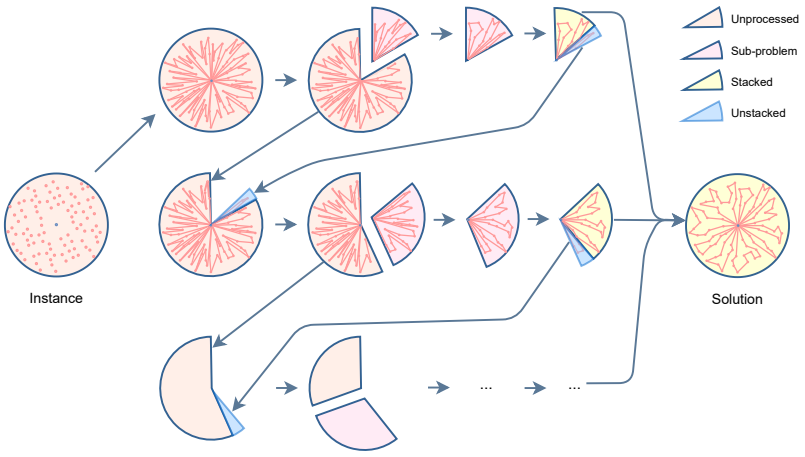



Figure 1: **A visualization of the *Clockwise Clustering* framework:** The clock-hand initializer is used to construct an initial solution. At each round, a small proportion of the tours are selected to form a sub-problem and sent to *Imitation Improvement Learning* to improve. A majority of the output are kept in a *sub-solution buffer* (*stacked*) and the remaining portion (*unstacked*) is merged with unprocessed tours to start the next round. Once all nodes are processed, we get all the sub-solutions from the sub-solution buffer and build a complete solution, save it and start a new round with the new solution to further improve it.

## 5 IMITATION IMPROVEMENT LEARNING

In this section, we present our *Imitation Improvement Learning* (IIL) framework, a main component of the *Clockwise Clustering* for solving sub-instances. Figure 2 provides an overview of the framework. Our framework is iterative in nature; a sub-solution $S$ is improved after each cycle loop. Starting with an initial sub-solution as $S$, a state $s(\mathcal{G}, S, \mathbb{B}, C)$ is forwarded to a neural encoder-decoder network to approximate the stochastic policy $\pi_\theta(a|s)$, where $\theta$ are trainable parameters. The value function $V_\phi(s)$ is also a neural network, where $\phi$ are trainable parameters. This platform uses policy gradient to optimize the parameters of the policy and value functions of the RL network. Here, thanks to the *Clockwise Clustering* platform, the sub-instances sent to IIL are significantly smaller in size, as compared to the original instance and have similar distributions. This matches the ability of learning-based RL models to produce high-quality solutions for small-sized instances (Chen & Tian, 2019; Hottung & Tierney, 2019). Intuitively, a good RL policy model would provide the heuristics with better local search space, leading to better expert solutions to further teach and improve the RL through imitation learning.

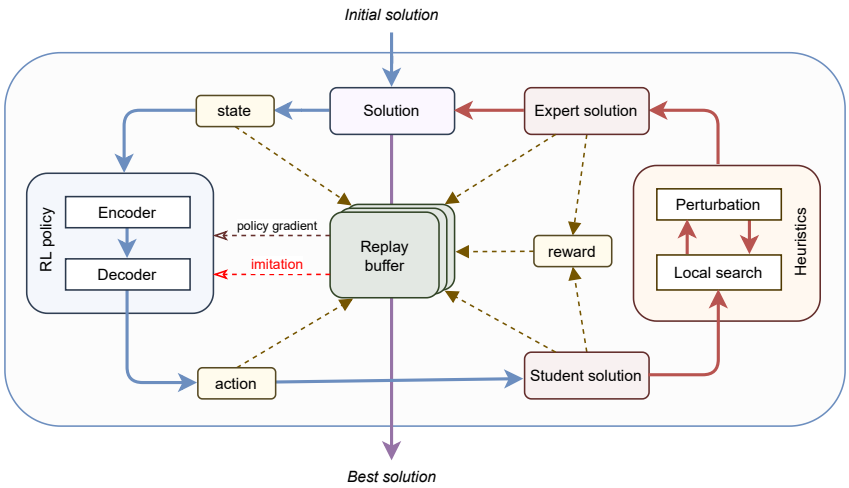



Figure 2: **An overview of our *Imitation Improvement Learning* framework**: Starting with an initial sub-solution $S$, the input is forwarded to an imitation cycle loop between RL Policy (student) and Heuristics (expert) to be improved after each iteration. At each iteration, the RL Policy, together with $k$-opt operators, generate a student solution and brings it directly to the traditional heuristics's local search to produce an expert solution. These two solutions are used to calculate rewards and imitation loss for training RL policy with policy gradient.

### 5.1 IMITATION LEARNING WITH EXPERTS

The key component of the IIL framework is an imitation learning model that use heuristics' solutions to teach the RL policy to generate high-quality solutions. More precisely, the RL policy model acts as a student who wants to learn from experts. Classical heuristics are ideal experts, which are also iterative and are able to generate good solutions quickly. Combining the loop of RL policy (student) and heuristics (expert), we have a closed loop where the imitation learning model encourages the RL policy to mimic the heuristics' policy. To train the imitation learning model, we employ the *generative adversarial imitation learning* (GAIL) algorithm (Ho & Ermon, 2016), which is based on a discriminative neural model $D_\delta$ to distinguish between solutions generated by the RL policy and the expert's (i.e. heuristics) policy, where $\delta$ are trainable parameters. The objective of the GAIL is $\max_\pi \min_\delta \mathbb{E}_\pi [\log(D_\delta(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D_\delta(s,a))]$, where $\pi_E$ refers to the experts' policy. By solving the max-min problem, one can force the RL policy to generate solutions that are similar to those generated by the expert's policy.

### 5.2 IMPROVEMENT LEARNING WITH $k$-OPT MOVES

Inspired by the *Learning 2-opt* algorithm (da Costa et al., 2021), our framework aims to select $k$ different nodes and swap its edges to form a student solution. The RL policy samples an action $a$ which contains $k$ nodes in $\mathbb{I}$ used for $k$-opt moves to generate a student solution $S_{student}$. After that, this solution is forwarded to the *Local Search* component of classical heuristics to get an expert solution $S_{expert}$. The heuristics don't need to be refreshed after generating this solution. Expert solution $S_{expert}$ continues to be forwarded to RL policy for a new loop. Rewards are computed by the different cost values of input and output solutions. For each step, replay buffer $\mathcal{D}$ collects states, actions, rewards and expert/student solutions for updating model parameters via policy gradient with RL loss and imitation learning loss.

### 5.3 NETWORK ARCHITECTURE

Our neural network is based on an encoder-decoder architecture (a detailed description is given in appendix). The encoder learns representations that embed graph topology. We create node features $\boldsymbol{X} \in \mathbb{R}^{|\mathbb{V}|\times 3}$ (x-coordinate, y-coordinate, and demand rate), edge features $\boldsymbol{E} \in \mathbb{R}^{|\mathbb{A}|\times 2}$ (euclidean distance and radian angle between each edge and x-axis), and import them into the Residual E-GAT (Lei et al., 2021) for feature extraction. Given these representations, the policy decoder samples action indices $a_1, a_2, ..., a_k$ sequentially for $k$-opt. We aim to learn the parameters of a stochastic policy $\pi_\theta(a|s)$ that, given a state $s$, assigns high probabilities to moves that reduce the cost of a tour. Our architecture uses a chain rule to factorize the probability of a $k$-opt move as $\pi_\theta(a|s) = \prod_{i=1}^{k} p_\theta(a_i|s)$. We use a pointing mechanism (da Costa et al., 2021) to predict a distribution over node outputs given encoded actions (nodes) and a state representation (query vector). The value decoder operates on the same encoder outputs but outputs real-valued estimates of state values. We give more details of network architecture in Appendix A.3.

### 5.4 LOSS FUNCTION

We use PPO (Schulman et al., 2017) for policy gradient optimization with the loss function $\mathbb{E}_{\pi_\theta}[\mathcal{L}^{PPO}_{\pi_\theta}]$. We also add an imitation loss to for imitation learning, leading to the following overall loss

$$\mathcal{L}^{IIL}_{\pi_\theta} = \mathcal{L}^{PPO}_{\pi_\theta} + c_{IL}\left\{\min_{\delta} \mathbb{E}_{\pi_\theta}\left[\log\left(D_\delta\left(s,a\right)\right)\right] + \mathbb{E}_{\pi_E}\left[\log\left(1 - D_\delta\left(s,a\right)\right)\right]\right\}$$

Where the first term is the PPO loss from standard RL policy and the second term (with weight parameter $c_{IL}$) is from the imitation learning model. By optimizing $\max_\theta \mathbb{E}[\mathcal{L}^{IIL}_{\pi_\theta}]$, we seek for a policy that both maximizes the standard long-term reward function of the RL policy and mimics the heuristics' policy. Intuitively, the first term of the loss function is to encourage exploration of new solutions and the second term is to exploit high-quality solutions from heuristics.

## 6 EXPERIMENTS AND RESULTS

We provide extensive experimental results based on some large-scale well-known CVRP datasets, targeting the following questions.

(i) By using heuristic methods as an expert/teacher for the policy model, can our *Imitation Improvement Learning framework* outperform the standing-alone heuristics?

(ii) Can the *Clockwise Clustering* and *Imitation Improvement Learning* frameworks help us solve very-large-scale instances with competitive performance?

(iii) Can our frameworks *generalize* to other instance sets of similar distributions?

(iv) Can our algorithms offer new *state-of-the-art (SOTA)* solutions for some popular CVRP instances?

Below, we present our datasets and our comparison results. Other details can be found in the appendix.

### 6.1 DATASET

We benchmark our frameworks using large-scale instances from three recent datasets from CVRPLIB (`http://vrp.atd-lab.inf.puc-rio.br/index.php/en/`), which is known to be challenging for both heuristic and learning-based methods. To test the generalizability of our clockwise clustering (CC) and IIL method, we experiment on an uniform dataset from Li et al. (2021). We also benchmark on *constrained electrical vehicle routing* (CEVRP) datasets, a CVRP variant for electrical vehicles with battery constraints. For this, Mavrovouniotis et al. (2020) contribute a large-scale CEVRP dataset containing 17 instances. Same as CVRP, we test the generalizability of our approaches in CEVRP by using this dataset to train the RL policy and uniformly generating 238 new instances to serve as a test set.

Six datasets used for benchmarking are listed in Table 1. First, we try to test the efficiency of our algorithms, compared to the heuristic counterparts, using Dataset 1. Next, we test the generalizability of our model by learning with a train set and evaluating with another test set of Dataset 2. We then benchmark our algorithms with two real-world large-scale datasets, called as Dataset 3&4, which are collected in Brazil and Belgium, respectively. Although most of the CVRP datasets use 2D euclidean distances, the DIMACS dataset (Dataset 3) use weights specifically defined for pairs of nodes. For CEVRP, we benchmark with instances from Mavrovouniotis et al. (2020) (named as Dataset 5) and our newly generated instances (Dataset 6).

Table 1: Datasets.

| Dataset | Source | VRP Type | Distance Space | # instances | # customers |
|---|---|---|---|---|---|
| 1 | Uchoa et al. (2017) | CVRP | Euclidean | 100 | 100-1000 |
| 2 | Li et al. (2021) | CVRP | Euclidean | 2000/40/40 | 500-2000 |
| 3 | DIMACS[1] | CVRP | Explicit | 12 | 241-1000 |
| 4 | Arnold et al. (2019) | CVRP | Euclidean | 10 | 3000-30000 |
| 5 | Mavrovouniotis et al. (2020) | CEVRP | Euclidean | 17 | 21-1000 |
| 6 | Ours | CEVRP | Euclidean | 238 | 21-1000 |

### 6.2 EXPERIMENTAL RESULTS AND ANALYSIS

With Dataset 1, we compare our methods with OR-tools, VNS, HILS Subramanian et al. (2013), KGLS Arnold & Sörensen (2019), SISR Christiaens & Berghe (2020), and HGS Vidal (2022). Instead of running HGS with full settings, we compare with HGS 30k steps and HGS $95\%$ solution quality, similarly to Li et al. (2021). Table 2 reports the results from previous works and our results, compared to the best known solutions (BKS) by GAP (percentage) scores, where IIL stands for our *Imitation Improvement Learning* method and RL stands for our learning-based framework but without the imitation learning loss. Our methods are clearly better than the corresponding standing-alone heuristics (i.e., RL+VNS and IIL+VNS versus VNS, and RL+HGS and IIL+HGS versus HGS). For instance, with HGS, our IIL method get $-0.27\%$ GAP vs BKS, better than $-0.30\%$ of HGS 30k steps. In terms of running time, our algorithms need about 10 hours while HGS takes about 16 to 40 hours to finish. We do not report the running times of the other methods because they are not reported in their respective papers, and these methods are clearly outperformed by our algorithms in terms of solution quality. The results also indicate that our IIL framework works the best with HGS sub-solver.

Table 3 shows a comparison of our best method IIL+HGS with previous results reported for Dataset 2 (Li et al., 2021). Note that all cost scores are divided by $1e5$. Our costs are better than the previous SOTA results obtained by Learning-to-delegate (L2D) (Li et al., 2021) for all the three sub-datasets. Although we use only one CPU worker without GPU for evaluating but the running time of our algorithm is just slightly longer than L2D (short).

Table 4 shows the results for Dataset 3 & 4. For Dataset 3, the classical HGS's score is slightly better than that of the IIL+HGS. Note that it's a non-euclidean dataset so we are not able to set correctly the node and edge features for our E-GAT encoder. It might affect the performance of our framework.

[1]The 12th DIMACS Implementation Challenge (`http://dimacs.rutgers.edu/programs/challenge/vrp/cvrp/cvrp-competition`)

Table 2: Experimental results for Dataset 1.

| Method | GAP vs BKS |
|---|---|
| OR-tools | -4.01% |
| VNS | -3.08% |
| HILS | -1.00% |
| KGLS | -0.66% |
| SISR | -0.54% |
| HGS (30k) | -0.30% |
| HGS (95%) | -0.48% |
| RL+VNS (ours) | -2.15% |
| RL+HGS (ours) | -0.31% |
| IIL+VNS (ours) | -1.79% |
| IIL+HGS (ours) | **-0.27%** |

Table 3: Experimental results for Dataset 2.

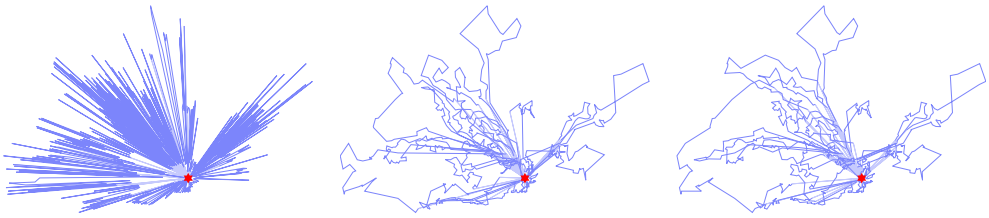
| Method | N=500 | | N=1000 | | N=2000 | |
|---|---|---|---|---|---|---|
| | Cost | Time | Cost | Time | Cost | Time |
| LKH-3 (95%) | 62.0 | 4.4min | 120.02 | 18min | 234.89 | 52min |
| LKH-3 (30k) | 61.87 | 30min | 119.88 | 77min | 234.65 | 149min |
| OR-tools | 65.59 | 15min | 126.52 | 15min | 244.65 | 15min |
| AM sampling | 69.08 | 4.70s | 151.01 | 17.40s | 356.69 | 32.29s |
| AM greedy | 68.58 | 25ms | 142.84 | 56ms | 307.86 | 147ms |
| NeuRewriter | 73.6 | 58s | 136.29 | 2.3min | 257.61 | 8.1min |
| Random | 61.99 | 71s | 120.02 | 3.2min | 234.88 | 6.4min |
| Count-based | 61.99 | 59s | 120.02 | 2.1min | 234.88 | 5.3min |
| Max Min | 61.99 | 59s | 120.02 | 2.5min | 234.89 | 5.2min |
| L2D (short) | 61.99 | 38s | 119.87 | 1.5min | 234.89 | 3.4min |
| L2D (long) | 61.7 | 76s | 119.55 | 3.0min | 233.86 | 6.8min |
| IIL+HGS (ours) | **60.49** | 68s | **118.37** | 2.6min | **225.43** | 6.3min |

In addition, initial solutions created by our *clock-hand* could be worst, making it difficult for our *Clockwise Clustering* framework to split nodes into sub-instances. Nevertheless, we achieved a new SOTA result for *Loggi-n501-k24* instance; our solution cost is 177078, better than the previous one 177176 reported by the CVRPLIB authors (`http://vrp.galgos.inf.puc-rio.br/index.php/en/updates/`). For Dataset 4, even-though it contains very-large-scale instances of sizes up to 30k, our IIL+HGS performs better than the classical HGS and any other methods. We illustrate initial solutions (from the Clock-hand initializer) and our best solutions for two large instances from Dataset 3&4 in Figure 3 and 4.

Table 4: GAP vs BKS (percentage) for two real-world datasets.

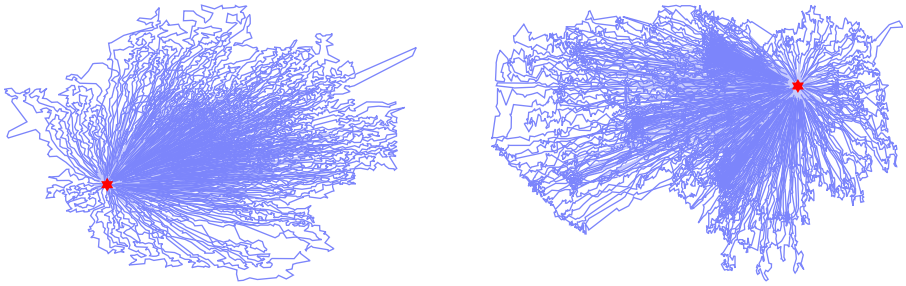
| Methods | Dataset 3 | Dataset 4 |
|---|---|---|
| CW | - | -8.00% |
| MDM | **-0.05%** | -3.63% |
| HGS | **-0.05%** | -4.89% |
| IIL+VNS (ours) | -0.77% | -6.71% |
| IIL+HGS (ours) | -0.09% | **-2.95%** |

Table 5 reports results for large-scale CEVRP instances from Dataset 5& 6, where "-" indicates that the information is not available from previous works. For these instances, battery information is added to the node features. IIL+HGS outperforms other classical heuristics. Specifically, IIL+HGS gets a new SOTA results with 1.88% GAP vs BKS. Note that HGS does not support the CEVRP variant with battery constraints, so we are not able to embed directly HGS library to our IIL framework. To make it work, we add a post-processing step, similarly to the GRASP (Woller et al., 2020), to rebuild feasible solutions that fit the battery constraints. To test the generalizability of our ap-



Initial solution - cost: 1406303 IIL+VNS - cost: 288820 IIL+HGS - cost: 285021

Figure 3: Instance *Loggi-n1001-k31* from *Dataset 3*, which does not support 2D Euclidean space.



Brussels Flanders

Figure 4: Solutions generated by IIL+HGS for two very-large-scale instances of *Dataset 4*, which represent the real-world maps of Brussels & Flanders, Belgium with sizes up to 30k.

proach, we run our IIL algorithms on Dataset 6 and compare the results with VNS and BACO (i.e., SOTA algorithms for the CEVRP instances), which clearly shows that IIL+HGS is much better than the heuristics in terms of both solution quality and running time.

Table 5: Experimental results for CEVRP datasets

| **Methods** | **Dataset 5** | | **Dataset 6** | |
|---|---|---|---|---|
| | GAP vs BKS | Time | GAP vs BKS | Time |
| GA | -4.57% | - | - | - |
| SA | -2.65% | - | - | - |
| VNS | -1.08% | 5.5h | 0.00% | 77.7h |
| BACO | -0.43% | 16.7h | 0.13% | 113.2h |
| IIL+VNS (ours) | 0.43% | 6.1h | 1.36% | 39.2h |
| IIL+HGS (ours) | **1.88%** | 2.5h | **2.15%** | 31.3h |

## 7 Conclusion

We proposed a new learning-based framework for CVRP, which employs heuristic methods as an expert to teach the RL policy model to generate high-quality solutions. To enhance the scalability and take the advantage of the RL approaches in learning from similar instances, we propose the *Clockwise Clustering* framework that offers good initial solutions and a nature way to decompose the whole instances into sub-instances of similar distributions. We benchmarked on several popular large-scale CVRP instances of sizes up to 30k, showing that our proposed algorithms outperform the respective standing-alone heuristics and offer competitive solutions, compared to previous SOTA algorithms. Our methods also archive new best solutions for several instances and generalize for a wide range of CVRP distributions and solvers.

Our work highlights the effectiveness of using generative adversarial imitation learning to help RL and heuristic methods work together in an iterative manner. An interesting direction for future work may extend our frameworks to other challenging combinatorial optimization problems.

## REFERENCES


Francesco Alesiani, Gülcin Ermis, and Konstantinos Gkiotsalitis. Constrained clustering for the capacitated vehicle routing problem (cc-cvrp). *Applied Artificial Intelligence*, 2022.

Florian Arnold and Kenneth Sörensen. What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Comput. Oper. Res.*, 106:280–288, 2019.

Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Comput. Oper. Res.*, 107:32–42, 2019.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, abs/1611.09940, 2017.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *NeurIPS*, 2019.

Johan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transp. Sci.*, 54:417–433, 2020.

Yves Crama, Antoon W. J. Kolen, and Erwin Pesch. Local search in combinatorial optimization. In *Artificial Neural Networks*, 1995.

G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6:791–812, 1958.

Paulo da Costa, Jason Rhuggenaath, Yingqian Zhang, Alp Eren Akçay, and Uzay Kaymak. Learning 2-opt heuristics for routing problems via deep reinforcement learning. *SN Comput. Sci.*, 2:388, 2021.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. doi: 10.1287/mnsc.6.1.80.

Bruce L. Golden, Sriram Raghavan, and Edward A. Wasil. The vehicle routing problem : latest advances and new challenges. 2008.

Pierre Hansen and Nenad Mladenović. First vs. best improvement: An empirical study. *Discret. Appl. Math.*, 154:802–817, 2006.

Dorothea Heiss-Czedik. An introduction to genetic algorithms. *Artificial Life*, 3:63–65, 1997.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. 12 2017. doi: 10.13140/RG.2.2.25569.40807.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.

Elias Boutros Khalil, Hanjun Dai, Yuyu Zhang, Bistra N. Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NIPS*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *ICLR*, 2019.

Suresh Kumar and Ramasamy Panneerselvam. A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 4:66–74, 2012.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

Gilbert Laporte. Fifty years of vehicle routing. *Transportation science*, 43(4):408–416, 2009.

Kun Lei, Peng Guo, Yi Wang, Xiao Wu, and Wenchao Zhao. Solve routing problems with a residual edge-graph attention neural network. *ArXiv*, abs/2105.02730, 2021.

Jan Karel Lenstra and Alexander H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *NeurIPS*, 2021.

S. Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.*, 21:498–516, 1973.

Manuel López-Ibáñez. Ant colony optimization. In *GECCO '10*, 2010.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

Michalis Mavrovouniotis, Charalambos Menelaou, Stelios Timotheou, Christos G. Panayiotou, Georgios Ellinas, and Marios M. Polycarpou. Benchmark set for the ieee wcci-2020 competition on evolutionary computation for the electric vehicle routing problem. 2020.

M. Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, 2018.

Marcelo Prates, Pedro Avelar, Henrique Lemos, Luís Lamb, and Moshe Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4731–4738, 07 2019. doi: 10.1609/aaai.v33i01.33014731.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Mohamed A. Wahby Shalaby, Ayman R. Mohammed, and Sally S. Kassem. Supervised fuzzy c-means techniques to solve the capacitated vehicle routing problem. *Int. Arab J. Inf. Technol.*, 18: 452–463, 2021.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael Maher and Jean-Francois Puget (eds.), *Principles and Practice of Constraint Programming — CP98*, pp. 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49481-2.

Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Comput. Oper. Res.*, 40:2519–2531, 2013.

Alex Syrichas and Alan Crispin. Large-scale vehicle routing problems: Quantum annealing, tunings and results. *Comput. Oper. Res.*, 87:52–62, 2017.

Éric D. Taillard, Luca Maria Gambardella, Michel Gendreau, and Jean-Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. *Eur. J. Oper. Res.*, 135:1–16, 2001.

Paolo Toth and Daniele Vigo. Vehicle routing: Problems, methods, and applications, second edition. 2014.

Wei Tu, Qingquan Li, Qiuping Li, Jiasong Zhu, Baoding Zhou, and Bi Yu Chen. A spatial parallel heuristic approach for solving very large-scale vehicle routing problems. *Transactions in GIS*, 21: 1130 – 1147, 2017.

Eduardo Uchoa, Diego Pecin, Artur Alves Pessoa, Marcus Poggi de Aragão, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *Eur. J. Oper. Res.*, 257:845–858, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio', and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2018.

Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2021.105643.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Oper. Res.*, 60:611–624, 2012.

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. Curran Associates, Inc., 2015.

Christos Voudouris and Edward P. K. Tsang. *Guided Local Search*, pp. 185–218. Springer US, Boston, MA, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/0-306-48056-5_7. URL https://doi.org/10.1007/0-306-48056-5_7.

David Woller, Viktor Kozák, and Miroslav Kulich. The grasp metaheuristic for the electric vehicle routing problem. In *MESAS*, 2020.

Cathy Wu, Kalyanaraman Shankari, Ece Kamar, Randy H. Katz, David E. Culler, Christos H. Papadimitriou, Eric Horvitz, and Alexandre M. Bayen. Optimizing the diamond lane: A more tractable carpool problem and algorithms. *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1389–1396, 2016.

# A Appendix

## A.1 Mixed-integer programming model for the CVRP

With a fully connected weighted graph $\mathcal{G} = (\mathbb{V}, \mathbb{A})$, the CVRP can be mathematically formulated as follows:

$$\min \sum_{i \in \mathbb{V}, j \in \mathbb{V}, i \neq j} d_{ij} x_{ij}, \tag{1}$$

s.t

$$\sum_{j \in \mathbb{V}, i \neq j} x_{ij} = 1, \forall i \in \mathbb{I}, \tag{2}$$

$$\sum_{j \in \mathbb{V}, i \neq j} x_{ij} - \sum_{j \in \mathbb{V}, i \neq j} x_{ji} = 0, \forall i \in \mathbb{V}, \tag{3}$$

$$u_j \leq u_i - b_i x_{ij} + C\left(1 - x_{ij}\right), \forall i \in \mathbb{V}, \forall j \in \mathbb{V}, i \neq j, \tag{4}$$

$$0 \leq u_i \leq C, i \in \mathbb{V}, \tag{5}$$

$$x_{ij} \in \{0, 1\}, i \in \mathbb{V}, j \in \mathbb{V}, i \neq j \tag{6}$$

where equation 1 defines the CVRP objective function, equation 2 ensures every customer is entered once, equation 3 establishes flow conservation by guaranteeing the number of times a vehicle enters a node is equal to the number of times it leaves that node. equation 4 and equation 5 guarantee demand fulfillment at all customers by assuring a non-negative carrying load upon arrival at any node including the depot, and equation 6 define a set of binary decision variables which each one equal to 1 if an arc is traveled and 0 otherwise. Variables $u_i$ denote the remaining carrying capacity of a vehicle on its arrival at node $i \in \mathbb{V}$.

## A.2 Algorithms

The pseudo-code of the *Clockwise Clustering* framework is provided in Algorithm 1. We first employ Clock-hand initializer for creating initial solution $S$ and send it into a loop to find the best

improvement solution $S^*$. Inspired by the *Divide-and-Conquer* mechanism, we select the first $u$ tours of $S$ to be a sub-problem $S_{sub}$, forward it to the *Imitation Improvement Learning* framework to find its optimal solution $S^*_{sub}$ and select its first $v$ tours for a solution buffer $S'$. After that, we combine all unprocessed nodes of $\mathcal{G}$ into $S$ and continue find improved solution until the stack $S'$ is full. Section A.4 below indicates how we specify $u$ and $v$ in our experiment settings. Figure 5 also provides an overview of our frameworks, showing how the *Clockwise Clustering* and *Imitation Improvement Learning* frameworks work together. Detailed steps of the *Imitation Improvement Learning* framework are provided in Algorithm 2.

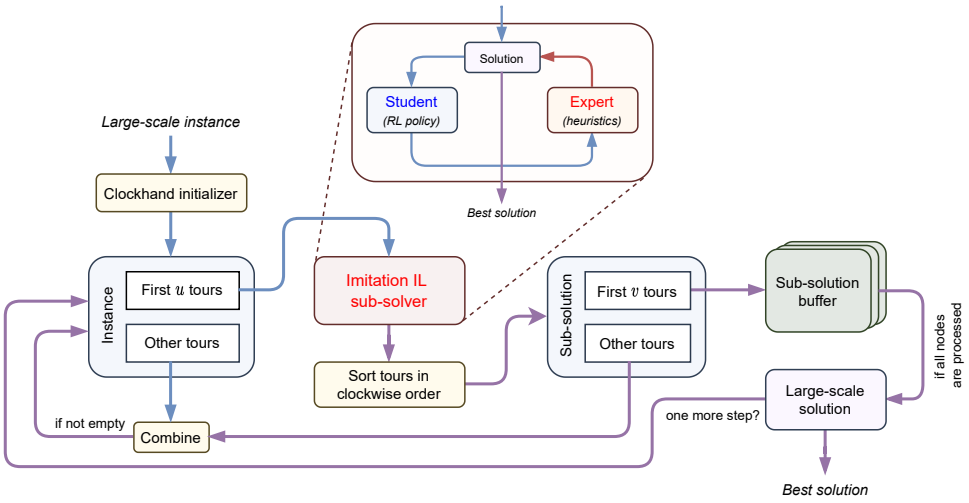



Figure 5: An overview of our *Clockwise Clustering* platform.

**Algorithm 1** Clockwise Clustering

$S \leftarrow \text{ClockHandInitializer}(\mathcal{G}, \mathbb{B}, C)$ ▷ Initial solution for $\mathcal{G}$
$S^* \leftarrow S$ ▷ Best solution
$S' \leftarrow []$ ▷ Solution buffer for $\mathcal{G}$
**while** repeat a certain number of times **do**
  $S_{sub} \leftarrow S_{:u}, S_{other} \leftarrow S_{u:}$ ▷ Select first $u$ tours for sub-problem
  $S^*_{sub} \leftarrow \text{IILSolver}(S_{sub}, \mathbb{B}, C)$
  $S'_{sub} \leftarrow S^*_{sub,:v}, S'_{other} \leftarrow S^*_{sub,v:}$ ▷ Select first $v$ tours for $\mathcal{G}$ solution
  $S' \leftarrow S' \oplus S'_{sub}$
  $S \leftarrow S_{other} \oplus S'_{other}$ ▷ Combine all unprocessed nodes
  **if** $S$ is empty **then** ▷ All nodes of $\mathcal{G}$ are processed
    **if** $\text{Cost}(S') < \text{Cost}(S^*)$ **then**
      $S^* \leftarrow S'$ ▷ Best solution ever
    **end if**
    $S \leftarrow S^*$
  **end if**
**end while**
**return** $S^*$

## A.3 Model Architecture

### A.3.1 Policy Encoder

With each input state $s(\mathcal{G}, S, \mathbb{B}, C)$, the encoder embeds the raw features of them into a higher-dimensional space. We create a node feature $\boldsymbol{x}_i \in \mathbb{R}^3$ by the values of x-coordinate, y-coordinate, and demand rate $\tilde{b}_i$ of each node $i$ in $\mathbb{V}$, where $\tilde{b}_i = b_i/C$. For each edge $(i, j)$ in $\mathbb{A}$, we also create

**Algorithm 2** Imitation Improvement Learning

$S \leftarrow \text{InitSolution}(\mathcal{G}, \mathbb{B}, C)$ ▷ Global solution
$\mathcal{D} \leftarrow []$ ▷ Replay buffer
**while** repeat a certain number of times **do** ▷ Number of global loops
  $s \leftarrow \text{InputFeatures}(S, \mathcal{G}, \mathbb{B}, C)$ ▷ State
  $a \leftarrow \text{SampleActionFromPolicy}(s)$ ▷ Action
  $S_{student} \leftarrow \text{ProcessOperators}(S, a)$ ▷ $k$-opt operation
  $S_h \leftarrow S_{student}$ ▷ Initiate heuristic solution
  **while** repeat a certain number of times **do** ▷ Number of heuristic loops
    $S'_h \leftarrow \text{LocalSearch}(S_h)$
    $S^*_h \leftarrow \text{Perturbation}(S'_h)$
    $S_h \leftarrow \text{AcceptanceCriterion}(S'_h, S^*_h)$
  **end while**
  $S_{expert} \leftarrow S_h$
  $r \leftarrow \text{Cost}(S_{student}) - \text{Cost}(S_{expert})$ ▷ Reward
  $\mathcal{D} \leftarrow \mathcal{D}.\text{append}((s, a, r, S_{student}, S_{expert}))$
  $S \leftarrow S_{expert}$ ▷ Update global solution
  **if** $|\mathcal{D}| > n_{\mathcal{D}}$ **then**
    $\text{UpdateRLWithImitation}(\mathcal{D})$
    $\mathcal{D} \leftarrow []$ ▷ Clear replay buffer
  **end if**
**end while**
**return** $S$

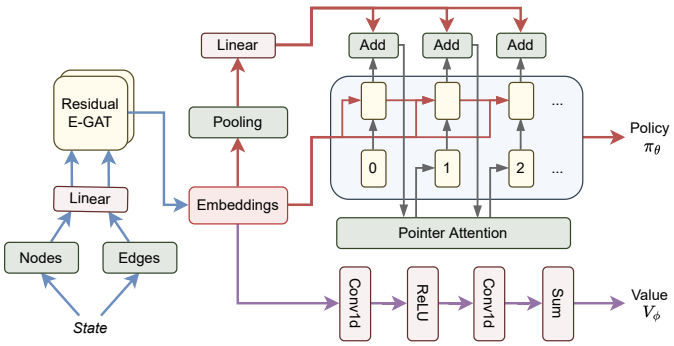



Figure 6: Model architecture of the RL policy.

an edge feature $\boldsymbol{e}_{ij} \in \mathbb{R}^2$ by the euclidean distance $d_{ij}$ and angle value $\theta_{ij}$ between node $i$ and $j$, where $\theta_{ij} = \arctan\left(\frac{y_i - y_j}{x_i - x_j}\right)$.

Both node and edge features are imported into Residual E-GAT (Lei et al., 2021) for feature extraction. Output of this layer is a set of representation vectors $\{\bar{\boldsymbol{x}}_i | i \in \mathbb{V}\}$, which embedded the information of $\mathcal{G}$, $\mathbb{B}$, and $C$. To represent the input solution $S$, we generate a context vector $H$ by mean pooling of each tour $t$ in $S$:

$$\boldsymbol{h} = \boldsymbol{W_h}\left(\frac{1}{|S|}\sum_{t \in S}\sum_{i \in t}\bar{\boldsymbol{x}}_i\right) + \boldsymbol{b_h},$$

where $\boldsymbol{W_h}$ and $\boldsymbol{b_h}$ are trainable parameters.

### A.3.2 Policy Decoder

We aim to learn the parameters of a stochastic policy $\pi_\theta(a|s)$ that, given a state $s$, assigns high probabilities to moves that reduce the cost of a tour. Our architecture uses a chain rule to factorize the probability of a $k$-opt move as $\pi_\theta(a|s) = \prod_{i=1}^{k} p_\theta(a_i|s)$. We use a pointing mechanism (da Costa

et al., 2021) to predict a distribution over node outputs given encoded actions (nodes) and a state representation (query vector).

Decoder predicts sequentially action $\hat{\pi}_t \in \mathbb{V}$ at time step $t \in \{1, ..., k\}$. Probability of action $\hat{\pi}_t$ is $p(\hat{\pi}_t|s) = \arg\max_{i \in \mathbb{V}} \frac{\exp(\boldsymbol{u}_{i,t})}{\sum_{j \in \mathbb{V}} \exp(\boldsymbol{u}_{j,t})}$, where $\boldsymbol{u}_t$ is the attention compatibility vectors of all nodes at time $t$. We set $\mathbb{M}_t$ is a set of different nodes allowed at time $t$. $\mathbb{M}_t = \{i | i \in S, i \neq \hat{\pi}_{t'} \forall t' < t\}$. Attention compatibility of node $i$ at time $t$ is

$$\boldsymbol{u}_{i,t} = \begin{cases} \tanh\left(\frac{\boldsymbol{Q}_{i,t}^T \boldsymbol{K}_i}{\sqrt{d}}\right) & \text{if } i \in \mathbb{M}_t \\ -\infty & \text{, otherwise} \end{cases},$$

where $\boldsymbol{K}$, $\boldsymbol{Q}$, and $d$ are key, query, and its dimension of this attention layer; $\boldsymbol{K}_i = \boldsymbol{W}_{\boldsymbol{K}} \bar{\boldsymbol{x}}_i$; $\boldsymbol{Q}_{i,t} = \text{MHA}\left(\boldsymbol{W}_{query} \boldsymbol{c}_t, \boldsymbol{W}_{key} \bar{\boldsymbol{x}}_i, \boldsymbol{W}_{value} \bar{\boldsymbol{x}}_i\right)$ is an output of a multi-head attention layer; context vector $\boldsymbol{c}_t = \boldsymbol{h} + \boldsymbol{W}_{\boldsymbol{c}} \bar{\boldsymbol{x}}_{\hat{\pi}_{t-1}} + \boldsymbol{b}_{\boldsymbol{c}}$, where $\boldsymbol{W}_{\boldsymbol{c}}$ and $\boldsymbol{b}_{\boldsymbol{c}}$ are trainable parameters. At time $t = 1$, $\hat{\pi}_{t-1}$ is set to the depot $0$.

### A.3.3 Value Decoder and Discrimination Model

Besides policy decoder, we also create neural networks for value decoder and discrimination model by using representation vectors from policy encoder. Value decoder $V_\phi(s)$ uses two convolution layers with ReLU and sum pooling as activation functions of each layer respectively:

$$V_\phi(s) = \sum_{i \in \mathbb{V}} \text{Conv}(\text{ReLU}(\text{Conv}(\bar{\boldsymbol{x}}_i)))$$

Discriminator $D_\delta(\tau)$ is a logistic classification model with a learned function $f_\delta$ predicts the probability of an input trajectory $\tau$ is student's or expert's. We uses multi-layer perceptron (MLP) with two fully connected layers for the discrimination network $D_\delta(\tau) = 1/(1 + \exp(-f_\delta(\tau))) = \sigma(\text{MLP}_\delta(\tau))$, where $\sigma$ is logistic sigmoid function.

## A.4 Hyperparameter Settings

All our experiments use the same set of hyperparameters given below.

**Network architecture:**

- Encoder: Two GATConv layers; hidden dim $128$.
- Decoder: $k$-opt with $k = 3$; using $8$ heads on multi-head attention layers; hidden dim $256$.

**PPO hyperparameters:**

- Clipping version, $\epsilon = 0.2$
- $\gamma = 0.995, \lambda = 0.95$
- Value coefficient $c_1 = 0.5$
- Entropy coefficient $c_2 = 0.01$
- Clipping gradient norm with max norm $0.5$ to avoid exploding gradients.

**Clockwise Cluster settings:**

- Initializer: Clockhand method
- Number of repeating solving a sub-problem: $128$ (VNS) or $24$ (HGS)
- Number of nodes collected for sub-problems of each loop: $1024$ (for very-large scale instance) or $256$ (otherwise)
- Number of the last tours removed in sub-problem solutions: $2$ (if number of tours is greater than $4$) or $0$ (otherwise)

**IIL settings:**

- Classical heuristic sub-solvers: VNS/HGS
- Number of training policy steps when the replay buffer is full: $64$
- Number of heuristic steps of each loop: $128$ (VNS) or $32$ (HGS)

**Training settings:**

- Adam optimization (Kingma & Ba, 2015) with weight decay (Loshchilov & Hutter, 2019) $\lambda = 0.01$
- Learning rate $lr =$ 3e-4
- Imitation coefficient $c_{IL} = 0.1$ (if use imitation loss) else $0.0$ (otherwise)
- Number of workers $n_{workers} = 32$
- Buffer size $n_{\mathcal{D}} = 16n_{workers}$
- Use CUDA for GPU acceleration
- Number of random seeds: 20

**Evaluating settings:**

- Number of workers $n_{workers} = 1$
- Use CPU only

**Device specifications:**

- CPU: Intel Xeon Silver xxx Processor, 24 cores, 48 threads
- GPU: NVIDIA GeForce RTX 3080 Ti 12GB
- Memory: 64GB of RAM

**Model configuration for experiments:**

- RL+VNS: $c_{IL} = 0.0$, VNS sub-solver
- RL+HGS: $c_{IL} = 0.0$, HGS sub-solver
- IIL+VNS: $c_{IL} = 0.1$, VNS sub-solver
- IIL+HGS: $c_{IL} = 0.1$, HGS sub-solver

### A.5 More experimental results

We give more details for the experiments reported in the main body of the paper.

Table 6: Experimental results with Dataset 3

| Instance | BKS | MDM | HGS | IIL+VNS (ours) | IIL+HGS (ours) |
|---|---|---|---|---|---|
| ORTEC-n242-k12 | 123750 | 123750 | 123750 | 124456 | 123808 |
| ORTEC-n323-k21 | 214071 | 214071 | 214071 | 215394 | 214125 |
| ORTEC-n405-k18 | 200986 | 200986 | 200986 | 202962 | 200986 |
| ORTEC-n455-k41 | 292485 | 292516 | 292516 | 294470 | 292516 |
| ORTEC-n510-k23 | 184529 | 184529 | 184529 | 185507 | 184746 |
| ORTEC-n701-k64 | 445543 | 445601 | 445541 | 450240 | 446537 |
| Loggi-n401-k23 | 336903 | 336963 | 337065 | 337497 | 337206 |
| Loggi-n501-k24 | 177176 | 177466 | 177428 | 178067 | 177078 |
| Loggi-n601-k19 | 113155 | 113174 | 113181 | 113707 | 113225 |
| Loggi-n601-k42 | 347059 | 347046 | 347052 | 349414 | 347082 |
| Loggi-n901-k42 | 246301 | 246360 | 246441 | 249671 | 247143 |
| Loggi-n1001-k31 | 284356 | 285521 | 285362 | 288820 | 285021 |
| GAP (vs BKS) | 0.00% | -0.05% | -0.05% | -0.77% | -0.09% |

Table 7: Experimental results with Dataset 4

| Instance | CW | MDM | HGS | IIL+VNS | IIL+HGS |
|---|---|---|---|---|---|
| Antwerp1 | 498422 | 486497 | 486959 | 490436 | 489367 |
| Antwerp2 | 322902 | 305039 | 305190 | 307696 | 299575 |
| Brussels1 | 532558 | 520643 | 527068 | 550138 | 522275 |
| Brussels2 | 386048 | 366817 | 368166 | 375128 | 360741 |
| Flanders1 | 7525575 | 7407580 | 7567917 | 8051525 | 7442533 |
| Flanders2 | 4805608 | 4603333 | 4950737 | 4921000 | 4567692 |
| Ghent1 | 490783 | 482315 | 485713 | 492605 | 482492 |
| Ghent2 | 287371 | 272117 | 272165 | 275536 | 264491 |
| Leuven1 | 200971 | 195102 | 194813 | 196665 | 195906 |
| Leuven2 | 125613 | 114412 | 114458 | 114708 | 112974 |
| GAP (vs BKS) | -8.00% | -3.63% | -4.89% | -6.71% | -2.95% |

Table 8: Experimental results with Dataset 5

| Instance | GA | SA | VNS | BACO | IIL+VNS | IIL+HGS |
|---|---|---|---|---|---|---|
| X-n143-k7 | 16489 | 16610 | 16028 | 15901 | 15884 | 15865 |
| X-n214-k11 | 11762 | 11404 | 11324 | 11133 | 11152 | 10992 |
| X-n351-k40 | 28008 | 27223 | 27065 | 26478 | 26654 | 26173 |
| X-n459-k26 | 26048 | 27223 | 25371 | 24764 | 24906 | 24263 |
| X-n573-k30 | 54190 | 51929 | 52182 | 53823 | 51663 | 51138 |
| X-n685-k75 | 73926 | 72550 | 71345 | 70835 | 70043 | 69071 |
| X-n749-k98 | 84035 | 81393 | 81002 | 80300 | 79794 | 78684 |
| X-n819-k171 | 170966 | 165070 | 164290 | 164721 | 161833 | 159396 |
| X-n916-k207 | 357392 | 342797 | 341650 | 342993 | 335912 | 334087 |
| X-n1001-k43 | 78833 | 78054 | 77476 | 76297 | 76145 | 74097 |
| GAP vs BKS | -4.57% | -2.65% | -1.08% | -0.43% | 0.43% | 1.88% |
| Running Time | - | - | 5.5h | 16.7h | 6.1h | 2.5h |

Table 9: Experimental results with Dataset 6

| Method | GAP vs Baseline | Running Time |
|---|---|---|
| VNS (baseline) | 0.00% | 77.7h |
| BACO | 0.13% | 113.2h |
| RL+VNS | 1.33% | 30.9h |
| RL+HGS | 2.10% | 26.2h |
| IRL+VNS | 1.36% | 39.2h |
| IRL+HGS | 2.15% | 31.3h |

Table 10: Detail scores for Dataset 3 (part I)

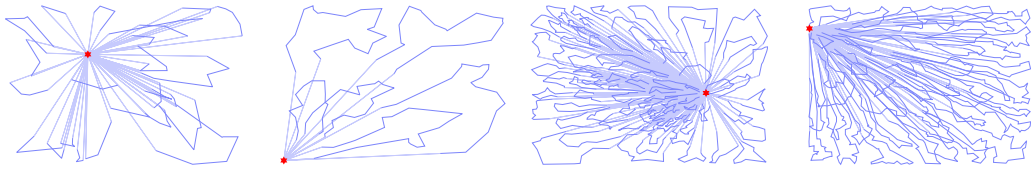
| Instance | VNS | OR-Tools | HGS (30k) | HGS (95%) | RL+ VNS | IIL+ VNS | RL+ HGS | IIL+ HGS |
|---|---|---|---|---|---|---|---|---|
| X-n101-k25 | 28588 | 27977 | 27591 | 27591 | 27858 | 28027 | 27591 | 27591 |
| X-n106-k14 | 26740 | 26758 | 26421 | 26421 | 26419 | 26583 | 26397 | 26375 |
| X-n110-k13 | 15264 | 15100 | 14971 | 14971 | 14985 | 15121 | 14971 | 14971 |
| X-n115-k10 | 12823 | 12808 | 12747 | 12747 | 12816 | 12823 | 12747 | 12747 |
| X-n120-k6 | 13746 | 13502 | 13332 | 13332 | 13429 | 13418 | 13332 | 13332 |
| X-n125-k30 | 56555 | 56853 | 55542 | 55546 | 56448 | 56348 | 55549 | 55542 |
| X-n129-k18 | 29709 | 29722 | 28940 | 28940 | 29007 | 29018 | 28940 | 28940 |
| X-n134-k13 | 11206 | 11171 | 10916 | 10916 | 11010 | 11003 | 10916 | 10916 |
| X-n139-k10 | 13793 | 13741 | 13590 | 13590 | 13704 | 13738 | 13590 | 13590 |
| X-n143-k7 | 16432 | 16136 | 15700 | 15700 | 15786 | 15879 | 15700 | 15700 |
| X-n148-k46 | 43986 | 44599 | 43448 | 43448 | 43962 | 43741 | 43448 | 43448 |
| X-n153-k22 | 22292 | 21789 | 21225 | 21225 | 22032 | 21589 | 21225 | 21225 |
| X-n157-k13 | 16972 | 17138 | 16876 | 16876 | 16894 | 16923 | 16876 | 16876 |
| X-n162-k11 | 14263 | 14262 | 14138 | 14138 | 14174 | 14179 | 14138 | 14138 |
| X-n167-k10 | 21143 | 21176 | 20557 | 20557 | 20969 | 20949 | 20557 | 20557 |
| X-n172-k51 | 46543 | 46875 | 45607 | 45607 | 46223 | 45812 | 45607 | 45607 |
| X-n176-k26 | 48971 | 49260 | 47812 | 47812 | 48888 | 48865 | 47815 | 47837 |
| X-n181-k23 | 25828 | 25936 | 25569 | 25579 | 25681 | 25637 | 25575 | 25575 |
| X-n186-k15 | 24787 | 24908 | 24145 | 24147 | 24662 | 24542 | 24145 | 24145 |
| X-n190-k8 | 17432 | 17422 | 17011 | 17039 | 17323 | 17276 | 17017 | 17029 |
| X-n195-k51 | 45297 | 46151 | 44225 | 44274 | 44767 | 44651 | 44225 | 44225 |
| X-n200-k36 | 60127 | 60448 | 58624 | 58632 | 60034 | 59952 | 58623 | 58608 |
| X-n204-k19 | 19960 | 20348 | 19570 | 19570 | 19820 | 19809 | 19566 | 19565 |
| X-n209-k16 | 31658 | 31776 | 30676 | 30680 | 31302 | 31141 | 30659 | 30683 |
| X-n214-k11 | 11296 | 11374 | 10880 | 10932 | 11180 | 11076 | 10873 | 10863 |
| X-n219-k73 | 117935 | 118038 | 117606 | 117626 | 117819 | 117700 | 117595 | 117610 |
| X-n223-k34 | 41540 | 42047 | 40437 | 40497 | 41135 | 41024 | 40496 | 40559 |
| X-n228-k23 | 26199 | 26613 | 25743 | 25745 | 26472 | 26325 | 25743 | 25745 |
| X-n233-k16 | 19833 | 19884 | 19230 | 19255 | 19588 | 19405 | 19230 | 19234 |
| X-n237-k14 | 27647 | 27928 | 27042 | 27042 | 27549 | 27437 | 27042 | 27044 |
| X-n242-k48 | 84081 | 85518 | 82846 | 83103 | 84164 | 83799 | 82898 | 83239 |
| X-n247-k50 | 38533 | 38283 | 37302 | 37377 | 38096 | 38075 | 37299 | 37300 |
| X-n251-k28 | 39552 | 40088 | 38805 | 38886 | 39418 | 39217 | 38724 | 38776 |
| X-n256-k16 | 19283 | 19295 | 18880 | 18880 | 19107 | 19120 | 18876 | 18880 |
| X-n261-k13 | 27352 | 27921 | 26621 | 26652 | 27170 | 27073 | 26627 | 26621 |
| X-n266-k58 | 77507 | 77661 | 75811 | 76122 | 76814 | 76751 | 75886 | 75867 |
| X-n270-k35 | 36111 | 36701 | 35303 | 35356 | 35935 | 35868 | 35319 | 35304 |
| X-n275-k28 | 21815 | 22087 | 21245 | 21271 | 21554 | 21453 | 21245 | 21250 |
| X-n280-k17 | 34659 | 35056 | 33558 | 33601 | 34320 | 34203 | 33551 | 33615 |
| X-n284-k15 | 20878 | 21138 | 20288 | 20299 | 20901 | 20730 | 20327 | 20317 |
| X-n289-k60 | 97897 | 98561 | 95530 | 95738 | 96986 | 96906 | 95665 | 95408 |
| X-n294-k50 | 48203 | 49302 | 47225 | 47247 | 48052 | 47755 | 47227 | 47222 |
| X-n298-k31 | 35532 | 36971 | 34291 | 34291 | 34985 | 35324 | 34240 | 34275 |
| X-n303-k21 | 22543 | 22574 | 21843 | 21867 | 22137 | 22081 | 21797 | 21763 |
| X-n308-k13 | 26605 | 27141 | 25891 | 25906 | 26426 | 26667 | 25890 | 25920 |
| X-n313-k71 | 96628 | 97497 | 94284 | 94458 | 95776 | 95709 | 94207 | 94090 |
| X-n317-k53 | 79129 | 79211 | 78400 | 78420 | 78671 | 78612 | 78400 | 78379 |
| X-n322-k28 | 30991 | 31489 | 29941 | 29965 | 30560 | 30324 | 29908 | 29834 |
| X-n327-k20 | 28867 | 28778 | 27582 | 27608 | 28450 | 28103 | 27587 | 27592 |
| X-n331-k15 | 32125 | 32648 | 31116 | 31150 | 31877 | 31623 | 31152 | 31105 |

Table 11: Detail scores for Dataset 3 (part II)

| Instance | VNS | OR-Tools | HGS (30k) | HGS (95%) | RL+ VNS | IIL+ VNS | RL+ HGS | IIL+ HGS |
|---|---|---|---|---|---|---|---|---|
| X-n336-k84 | 142258 | 143295 | 140048 | 140414 | 141908 | 141476 | 139671 | 139295 |
| X-n344-k43 | 43368 | 44036 | 42147 | 42147 | 43075 | 43041 | 42091 | 42080 |
| X-n351-k40 | 26578 | 27434 | 26023 | 26046 | 26594 | 26472 | 25990 | 25959 |
| X-n359-k29 | 53538 | 53858 | 51788 | 52080 | 52792 | 52652 | 51844 | 51727 |
| X-n367-k17 | 23987 | 23874 | 22814 | 22818 | 23605 | 23306 | 22828 | 22822 |
| X-n376-k94 | 148355 | 148776 | 147757 | 147946 | 148256 | 148054 | 147823 | 147749 |
| X-n384-k52 | 68423 | 69022 | 66118 | 66233 | 67711 | 67239 | 66171 | 66115 |
| X-n393-k38 | 39856 | 40786 | 38319 | 38527 | 39375 | 39131 | 38298 | 38373 |
| X-n401-k29 | 67771 | 68249 | 66365 | 66487 | 67478 | 67053 | 66450 | 66431 |
| X-n411-k19 | 20624 | 20811 | 19751 | 19797 | 20190 | 20095 | 19751 | 19736 |
| X-n420-k130 | 110505 | 111594 | 107937 | 108084 | 109704 | 109127 | 107917 | 107890 |
| X-n429-k61 | 67499 | 68858 | 65622 | 65810 | 67093 | 66602 | 65554 | 65518 |
| X-n439-k37 | 37074 | 37655 | 36430 | 36431 | 37063 | 36870 | 36417 | 36434 |
| X-n449-k29 | 58101 | 58427 | 55693 | 55852 | 57179 | 56968 | 55777 | 55562 |
| X-n459-k26 | 25535 | 25835 | 24211 | 24236 | 24921 | 24783 | 24176 | 24160 |
| X-n469-k138 | 230351 | 230963 | 223359 | 224071 | 227018 | 225629 | 222876 | 222550 |
| X-n480-k70 | 91887 | 92923 | 89775 | 89876 | 91496 | 90952 | 89691 | 89897 |
| X-n491-k59 | 68555 | 70817 | 66911 | 67208 | 68399 | 67880 | 67042 | 66802 |
| X-n502-k39 | 69971 | 70167 | 69272 | 69312 | 69680 | 69615 | 69276 | 69315 |
| X-n513-k21 | 25414 | 25846 | 24236 | 24236 | 24776 | 24731 | 24259 | 24281 |
| X-n524-k153 | 157354 | 156897 | 154898 | 155103 | 157963 | 157243 | 154714 | 154859 |
| X-n536-k96 | 98060 | 99576 | 95293 | 95653 | 97051 | 96391 | 95395 | 95141 |
| X-n548-k50 | 88752 | 89383 | 86974 | 87065 | 88131 | 87860 | 86883 | 87009 |
| X-n561-k42 | 44378 | 45759 | 42809 | 42829 | 44003 | 43931 | 42762 | 42819 |
| X-n573-k30 | 52059 | 52437 | 50909 | 51110 | 51405 | 51393 | 50969 | 50871 |
| X-n586-k159 | 197669 | 198347 | 191043 | 191890 | 193932 | 193017 | 190960 | 191168 |
| X-n599-k92 | 112460 | 113381 | 109467 | 109966 | 111220 | 110622 | 108933 | 108700 |
| X-n613-k62 | 62205 | 64074 | 59865 | 60105 | 61308 | 61045 | 59907 | 59735 |
| X-n627-k43 | 64805 | 64898 | 62558 | 62760 | 64502 | 64064 | 62704 | 62561 |
| X-n641-k35 | 66801 | 66862 | 64217 | 64485 | 66165 | 65965 | 64275 | 64015 |
| X-n655-k131 | 107469 | 107816 | 106857 | 106961 | 107367 | 107200 | 106925 | 106961 |
| X-n670-k130 | 150623 | 151874 | 146838 | 147390 | 152982 | 152381 | 146856 | 146720 |
| X-n685-k75 | 71130 | 74086 | 68502 | 68676 | 70005 | 69826 | 68536 | 68518 |
| X-n701-k44 | 85231 | 87060 | 82852 | 83175 | 84531 | 84458 | 82830 | 83026 |
| X-n716-k35 | 45519 | 46013 | 43640 | 43791 | 44990 | 44535 | 44032 | 43919 |
| X-n733-k159 | 139838 | 143829 | 136613 | 137173 | 139030 | 138475 | 136653 | 136841 |
| X-n749-k98 | 80269 | 82813 | 78113 | 78459 | 79526 | 79287 | 78162 | 77896 |
| X-n766-k71 | 119155 | 123106 | 114965 | 115425 | 118121 | 117951 | 115346 | 115237 |
| X-n783-k48 | 76475 | 77519 | 73457 | 73812 | 75216 | 74739 | 73232 | 73086 |
| X-n801-k40 | 76461 | 76428 | 73617 | 73934 | 75811 | 75076 | 73686 | 73703 |
| X-n819-k171 | 164244 | 165074 | 158870 | 159631 | 161557 | 161126 | 159037 | 158729 |
| X-n837-k142 | 199937 | 201837 | 195237 | 196195 | 197979 | 196878 | 195398 | 195419 |
| X-n856-k95 | 90906 | 91614 | 89060 | 89126 | 90240 | 89948 | 89085 | 89208 |
| X-n876-k59 | 102839 | 103576 | 100317 | 100769 | 101791 | 101339 | 100376 | 100433 |
| X-n895-k37 | 57365 | 58192 | 54320 | 54588 | 56280 | 56209 | 54308 | 54190 |
| X-n916-k207 | 340623 | 342127 | 331472 | 331993 | 335394 | 333588 | 331754 | 331691 |
| X-n936-k151 | 138629 | 140479 | 133426 | 133754 | 139309 | 137563 | 133279 | 133113 |
| X-n957-k87 | 88487 | 88603 | 85821 | 86086 | 87295 | 86765 | 85605 | 85757 |
| X-n979-k58 | 122827 | 123885 | 120070 | 120255 | 123360 | 122935 | 119927 | 119697 |
| X-n1001-k43 | 76363 | 78085 | 73509 | 74002 | 75588 | 75011 | 73411 | 73354 |

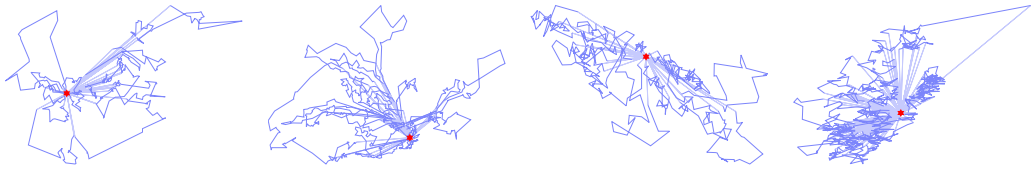
Table 12: Detail scores for Dataset 5.

| Instance | Index | VNS | BACO | RL+VNS | IIL+VNS | RL+HGS | IIL+HGS |
|---|---|---|---|---|---|---|---|
| X-n143-k7 | min | 16028.1 | 15901.2 | 15894.6 | 15883.7 | 15865.2 | 15865.2 |
| | mean | 16459.3 | 16031.5 | 15935.5 | 16068.3 | 15877.8 | 15878.6 |
| | std | 242.6 | 262.5 | 48.0 | 117.1 | 5.7 | 5.0 |
| X-n214-k11 | min | 11323.6 | 11133.1 | 11199.3 | 11152.3 | 11018.9 | 10991.8 |
| | mean | 11482.2 | 11219.7 | 11277.7 | 11274.2 | 11026.7 | 11025.6 |
| | std | 76.1 | 46.3 | 63.3 | 67.8 | 8.2 | 14.0 |
| X-n351-k40 | min | 27064.9 | 26478.3 | 26806.1 | 26653.6 | 26280.5 | 26172.8 |
| | mean | 27217.8 | 26593.2 | 26831.3 | 26792.2 | 26322.4 | 26271.0 |
| | std | 86.2 | 72.9 | 21.6 | 79.1 | 26.5 | 42.1 |
| X-n459-k26 | min | 25370.8 | 24763.9 | 25050.8 | 24905.7 | 24338.1 | 24263.3 |
| | mean | 25582.3 | 24916.6 | 25080.6 | 25046.9 | 24363.2 | 24357.2 |
| | std | 106.9 | 94.1 | 28.6 | 81.2 | 19.1 | 55.0 |
| X-n573-k30 | min | 52181.5 | 53822.9 | 51774.5 | 51662.6 | 51264.7 | 51138.1 |
| | mean | 52548.1 | 54567.2 | 51942.9 | 51861.3 | 51342.6 | 51361.4 |
| | std | 278.9 | 231.1 | 133.9 | 131.6 | 52.0 | 113.7 |
| X-n685-k75 | min | 71345.4 | 70834.9 | 70265.6 | 70043.2 | 69197.4 | 69071.4 |
| | mean | 71770.6 | 71440.6 | 70527.3 | 70415.3 | 69244.6 | 69185.4 |
| | std | 197.1 | 281.8 | 169.3 | 177.4 | 42.3 | 65.6 |
| X-n749-k98 | min | 81002.0 | 80299.8 | 79989.7 | 79794.3 | 79001.3 | 78683.9 |
| | mean | 81327.4 | 80694.5 | 80274.1 | 80074.2 | 79062.5 | 78899.5 |
| | std | 176.2 | 223.9 | 166.4 | 138.3 | 79.7 | 104.8 |
| X-n819-k171 | min | 164290.0 | 164720.8 | 162261.4 | 161832.7 | 159766.9 | 159396.0 |
| | mean | 81327.4 | 80694.5 | 162611.1 | 162093.5 | 159852.6 | 159607.8 |
| | std | 176.2 | 223.9 | 181.4 | 146.7 | 53.9 | 103.0 |
| X-n916-k207 | min | 341649.9 | 342993.0 | 337513.5 | 335911.8 | 333647.2 | 334086.8 |
| | mean | 342460.7 | 345000.0 | 337977.5 | 336633.8 | 333990.3 | 334554.1 |
| | std | 510.7 | 905.7 | 252.8 | 350.0 | 158.2 | 254.6 |
| X-n1001-k43 | min | 77476.4 | 76297.1 | 76543.9 | 76145.4 | 74231.4 | 74097.1 |
| | mean | 77920.5 | 77434.3 | 76778.8 | 76538.8 | 74485.5 | 74483.4 |
| | std | 234.7 | 719.9 | 127.4 | 257.1 | 161.4 | 167.9 |



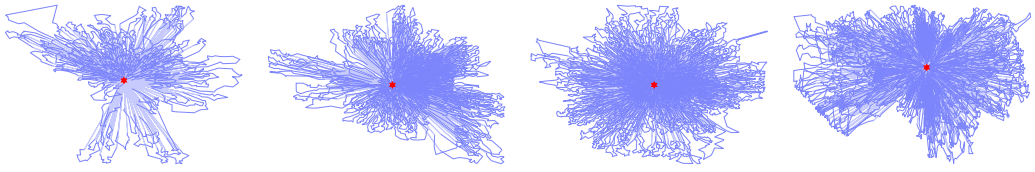
X-n101-k25 X-n120-k6 X-n957-k87 X-n1001-k43

(a) Dataset 1



Loggi-n501-k24 Loggi-n1001-k31 ORTEC-n405-k18 ORTEC-n701-k64

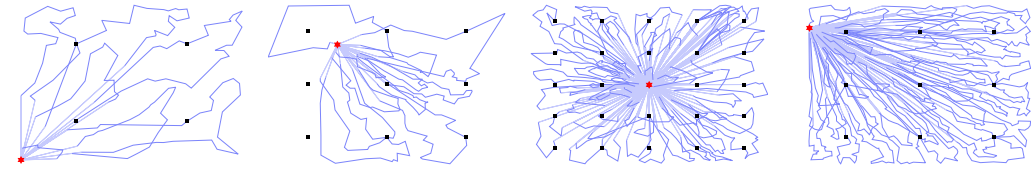
(b) Dataset 3



Leuven1 Ghent1 Brussels1 Flanders1

Leuven2 Ghent2 Brussels2 Flanders2

(c) Dataset 4



X-n143-k7 X-n214-k11 X-n685-k75 X-n1001-k43

(d) Dataset 5

Figure 7: Visualization of several optimal solutions found by our methods