RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning

Zefang Zong*
Department of Electronic Engineering, Tsinghua University
Beijing Tsingroc Co. Ltd
zongzf19@mails.tsinghua.edu.cn

Hansen Wang*
Institute of Interdisciplinary Sciences, Tsinghua University
139102262@qq.com

Jingwei Wang
Department of Electronic Engineering, Tsinghua University
jingwei-18@mails.tsinghua.edu.cn

Meng Zheng
Hitachi (China) Research & Development Corporation
mengzheng@hitachi.cn

Yong Li
Department of Electronic Engineering, Tsinghua University
liyong07@tsinghua.edu.cn

ABSTRACT

The large-scale vehicle routing problems (VRPs) are defined based on the classical VRPs with thousands of customers. It is an important optimization problem in modern logistic systems, since efficiently obtaining high-quality solutions can greatly reduce operation expenses as well as improve customer satisfaction. Most existing algorithms, including traditional non-learning heuristics and learning-based methods, only perform well on small-scale instances with usually no more than hundreds of customers. In this paper we present a novel Rewriting-by-Generating (RBG) framework which solves large-scale VRPs hierarchically. RBG consists of a rewriter agent that refines the customer division globally and an elementary generator to infer regional solutions locally. It is also flexible with multiple CVRP variant problems and could be continuously evolved with more up-to-date generator designs. We conduct extensive experiments on both synthetic and real-world data to demonstrate the effectiveness and efficiency of our proposed RBG framework. It outperforms HGS, one of the best heuristic method for CVRPs and also shortens the inference time. Online evaluation is also conducted on a deployed express platform in Guangdong, China, where RBG shows advantages to other alternative built-in algorithms.

CCS CONCEPTS

• **Applied computing** → **Transportation**.

KEYWORDS

Large Scale; Vehicle Routing Problems; Reinforcement Learning

*Equal Contribution.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539037

ACM Reference Format:
Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. 2022. RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539037

1 INTRODUCTION

The Vehicle Routing Problems (VRPs) is an important combinatorial optimization problem, which is a vital task in modern logistic system designs. Each vehicle must start from a depot and deliver parcels to customers, while each customer is serviced once and only once. Efficiently generating high-quality solutions can greatly reduce operation expenses of traveling cost due to the gasoline consumption. Moreover, with the service amounts growing continuously, practical logistic systems often have to handle with enormous distribution of customer nodes, usually more than several hundreds to thousands, which forms the *Large-Scale VRPs*. However, most existing works focus on finding near-optimal solutions with only no more than a hundred customers because of the computational complexity [8, 17, 22]. Due to the NP-hard nature, the exponential expansion of solution space makes it much more difficult than solving a small-scale one. Therefore, providing effective and efficient solutions for large-scale VRPs is a challenging problem.

Current algorithms proposed for VRPs can be divided into traditional non-learning based heuristics and reinforcement learning (RL) based models. Many routing solvers involve heuristics as their core algorithms, for instance, OR Tools [18], LKH3 [19] and HGS [33], which can find a near optimal solution by exploring the solution space using manually defined operations. However, pure heuristic based methods suffers from limitation of hand-crafted rules and computation burdens. Apart from traditional heuristics, RL based VRP solutions have been widely studied recently to find more efficient and effective solutions [7, 9, 10, 20, 25, 26]. Due to the learning manner that takes every feedback from learning attempts as signals, RL based methods rely on few hand-crafted rules and thus can be widely used in different customer distributions without human intervention and expert knowledge. However, current RL

agents are still insufficient to learn a feasible policy or generate solutions directly on large-scale VRPs due to the vast solution space, which grows exponentially along with customer size N. Consequently, the complexity makes the agent difficult to fully explore the space and makes the model hard to learn useful knowledge in large-scale VRPs.

To avoid the computation explosion in large-scale VRPs, we consider leveraging the classic Divide-and-Conquer idea to decompose the enormous scale of the original problem. In particular, we divide the large-scale customer distributions into small-scale ones and then generate individual regional solutions to reduce the problem complexity. However, how to 1) obtain a refined region division where the local VRPs can be handled effectively and how to 2) coordinate iterations between global and local optimization efficiently remain two challenges of our structure.

To tackle these two challenges above, we propose an RL-based hierarchical framework, named **Rewriting-by-Generating (RBG)**, to solve large-scale VRPs. The framework is consist of a "Generator" and a "Rewriter". First, we divide customers into regions and use an elementary VRP solvers to solve them locally, known as the "Generation" process. After that, from a global perspective, a special "Rewriting" process is designed based on all regional generations, which rewrites the previous solution with new divisions and the corresponding new regional VRP results. Within each rewriting step, we select and merge two regions into a hyper-region, and further divide it into two new sub-regions according to the hyper-regional VRP solution. By doing this, the problem scale is decomposed into pieces where it could be solved efficiently using regional solvers, and can still preserve the solution quality improved by the RL-based rewriter continuously.

To summarize, the main contributions of this paper are as follows,

- We formally address the large-scale vehicle routing problem and present an end-to-end framework, Rewriting-by-Generating (RBG) to solve it.
- We propose a hierarchical framework that consists of a "Generating" and a "Rewriting" process respectively. The framework has the potential to be continuously improved by augmenting new advanced solvers as its generator.
- Extensive experiments demonstrate that our RBG framework achieves significant performance in a much more efficient manner. It shows significant advantage on both solution quality and solving speed to other baselines. Moreover, it also shows growing superiority to other methods when the problem scale keeps increasing, and obtains great robustness to different data distributions.

The remainder of this paper is organized as follows. First, we introduce some related works on solving VRPs in Section 2. Then, we present our RBG framework in Section 3. The evaluation results are shown in Section 4. Lastly, we conclude this paper in Section 5.

2 RELATED WORK

2.1 Traditional Heuristics.

Due to the fact that a VRP instance has its own theoretical optimal solution, researchers attempted to develop exact methods as early solutions [4, 22]. However, these methods are only effective on small size scales and are extremely slow. Since the exact methods are almost impossible to solve VRPs within a reasonable time due to the high computation complexity, researchers developed heuristics as non-exact methods, to find approximation solutions instead. Tabu search is one of the old meta-heuristics [5, 15], which keeps searching for new solutions in the neighborhood of the current solution. Instead of focusing on improving merely one solution, genetic algorithms operate in a series of solutions [16, 27]. They construct new structures continuously based on parent structures. Instead of treating objectives to be optimized altogether, ant colony optimizations as another widely accepted solver utilize several colonies to optimize different functions: the number of vehicles, the total distance and others [12, 37]. Meanwhile, recreate search methods keep constructing the current solution and ruining the current ones to build better solutions. [29]. This helps to expand the exploration space to prevent the local optimum. Among these categories, LKH3 is one of the best and most widely-used heuristic solvers that empirically finds optimal solutions [19]. As a solution designed for large scale VRPs specifically, HGS [33] shows both its efficiency and effectiveness. However, the dependence on handcrafted rules limits the method in solving basic CVRPs only, and is hard to generalize to other routing variant problems.

Although these heuristics can improve searching efficiency compared to exact methods, they are still much too time-consuming when applied to large-scale VRPs with acceptable performances required, and are vital in responding to any real-time solution requests.

2.2 RL based VRPs Solutions.

Since the learning manner allows the agent model to directly infer solutions based on a pre-trained model with much shorter computation time, RL based methods become a compelling direction on solving combinatorial optimizations. It has been successfully applied in VRPs particularly [7, 20, 26]. Vinyals et al. [34] was the first to adopt deep learning in combinatorial optimizations by a novel *Pointer Network* model. Inspired by this, Bello et al. [7] proposed to use RL to generate routing solutions from scratch. Kool et al. [20] further augments the attention mechanism into the similar structure and solved more generalized combinatorial optimization problems. Other than using the idea of $PointerNetwork$, Dai et al. [10] develops their method over graphs via Q-learning [31], so that the solutions could have better generalization ability. Chen et al. [9] proposed a local rewriting rule that keeps rewriting the local components of the current situation via a Q-Actor-Critic training process [31]. Lu et al. [25] further developed a Learn-to-Iterate structure that not only improves the solution exploration but also generates perturbations to avoid local optimum. This is the first machine learning framework that outperforms LKH3 on CVRPs, both in computation time and solution quality.Delarue et al. [11] models the action selection space as a mixed integer optimization problem and use value-function-based RL to solve VRPs. Li et al. [23], as a homochronous work as ours tackles large scale VRPs specifically. Instead of conducting rewriting process on the division of local region, they choose to train a neural network to generate and select subproblems directly out of all routes from the last iteration globally.

However, these existing RL based methods only achieve promising results at small scales. The dataset evaluated in these works are usually constructed with $N = 20, 50, 100$ only. The proposed models cannot be trained for thousand-customer-level VRPs because the state space and action space extend exponentially as the number of customers increases, it will be hard for the model to learn useful route generation policy directly. In contrast, we propose a hierarchical RL based framework formed via the classical idea of Divide-and-Conquer to solve the large-scale challenge.

3 PRELIMINARY

Among all VRP variants, we mainly focus on the capacitated Vehicle Routing Problem (CVRP). CVRP involves one depot where vehicles start and end, and several customers with different demands. The task is to determine the routes with limited capacity to visit all the customers and fulfill their demands in order to minimize the total traveling distance.

Let $G(V, E)$ denote the graph consisting of depot and customers. Specially, $V = \{v_0, v_1, ..., v_i, ..., v_N\}$, where v_0 denotes the depot, and $v_i (1 \leq i \leq N)$ denotes the i-th customer with its location (x_i, y_i) and demand $d_i > 0$. For each pair of nodes v_i and v_j, the edge $e_{i,j}$, or $E(v_i, v_j)$ in another manner represents the distance between v_i and v_j. In accordance with previous paper[9, 20, 25], $e_{i,j}$ is set as the Euclidean distance of its two nodes, i.e., $e_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

The capacity of the vehicle is denoted by D, which is the maximal constraint to its loaded shipment. The condition $d_i \leq D$ is always satisfied so that the vehicle only visits a customer once in the entire solution. The vehicle must return to the depot v_0 to reload when the remaining goods can satisfy no more customers. Therefore, to meet the needs of every customer, the vehicle may return to the depot several times, and we term such a travelling sequence with its head and tail located at the depot v_0 as a *route*. We denote the k-th route τ_k as $\tau_k = (v_0, \tau_{k,1}, \tau_{k,2}, ..., \tau_{k,n}, v_0)$, where $\tau_{k,j} \neq v_0 (j = 1, ..., n_k)$ are the customers visited in this route in order and n_k is the number of them. The corresponding cost is $C(\tau_k) = E(v_0, \tau_{k,1}) + \sum_{j=1}^{n-1} E(\tau_{k,j}, \tau_{k,j+1}) + E(\tau_{k,n}, v_0)$. Thus the total solution π of the problem is composed of K routes $\pi = \{\tau_k, k = 1, ..., K\}$, and the total cost is $C(\pi) = \sum_{k=1}^{K} C(\tau_k)$.

With the above notations, we mathematically define CVRP as follows,

$$\min \sum_{k=1}^{K} C(\tau_k), \tag{1}$$

$$s.t. \quad \tau_1 \cup \tau_2 \cup \cdots \cup \tau_i \cdots \cup \tau_K = V, \tag{2}$$

$$\tau_i \cap \tau_j = \{v_0\}, \forall i \neq j, \tag{3}$$

$$\sum_{i=1}^{n_k} d_{\tau_{k,i}} \leq D, \forall 1 \leq k \leq K \tag{4}$$

$$\tau_{k,i} \neq v_0, \forall 1 \leq k \leq K, 1 \leq i \leq n_k \tag{5}$$

where constraint (2) and (3) ensure all customers visited and only visited once with demands satisfied. (4) indicates the capacity constraint. (5) is the constraint of routes that there is no intermediate visit of depot v_0 in a single route.

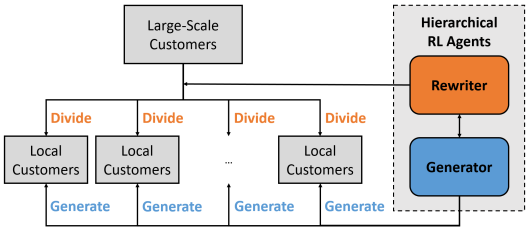


Figure 1: The overview of the Rewriting-by-Generating Framework.

4 REWRITING-BY-GENERATING

4.1 Overall Architecture

Figure 1 shows the overview structure of our proposed framework, named Rewriting-by-Generating (RBG). Along with the fundamental idea of Divide-and-Conquer to decompose the enormous problem scale as discussed previously, we aim at dividing the total customers into separate regions and generate near-optimal solutions individually. To achieve this, we design an RL-based hierarchical structure including two different components.

First, to refine and obtain more reasonable division results, we design the "Rewriting" process which keeps updating new divisions by rewriting the previous ones and their corresponding regional solutions. The division quality is critical to the final solution since customers from different regions cannot be scheduled upon the same route. Within each rewriting step, the agent selects and merges two regions based on their current local solutions. A new solution will be generated upon the merged hyper-region in the following step, and the rewriter will further divide the merged hyper-region back to two new regions. Since the exploration on different customer composition is complicated and it is not trivial to measure the direct influence to the final performance in terms of traveling distance, an RL-based rewriter is a wise choice to learn the selection and merging action which takes long-term reward as signals. We will show that the model converges and achieves high performance when the rewriter agent learns a stable division result in Section 4.

Second, to reach the global solution from the regional scratches, we employ an elementary VRP generator that generates solutions to each region, known as the "Generating" process. The selection of the generator is flexible, which enables the RBG to evolve with any up-to-date end-to-end VRP solvers and could generate to other practical routing problems with constraints by implementing specific routing solvers.

Overall, we develop an RL-based hierarchical framework by coordinating the rewriter and the generator in both global and local perspectives. The rewriter updates new division and brings new customer distributions to the generator globally, while the solutions from the generator formulate a key component of the rewriter in each local regions. From the technical perspective, it is worthy to note that the merging-repartitioning operation the rewriter conducts is also adopted in previous meta-heuristics [3, 6], while we replace the handcrafted heuristic with a learning agent. The global RL based rewriter is responsible for managing inter-regional

exploration while the generator optimizes inner-results. The combination of Operation Research (OR) heuristics and RL guarantees an effective exploration process as well as achieving high computation efficiency from the prospect of fast solution generation on inference instances.

For brevity and clarity, we summarize the pipeline as five steps, as shown in Figure 2. First, we cluster customers into several initialized hyper-regions. Second, we generate initial regional VRP solutions in individual hyper-regions via our elementary VRP generator. Third, we utilize the rewriter to partition each hyper-region into two smaller regions. Then the rewriter picks up two sub-regions via neural region representations and merges them into one hyper-region, and finally generates the hyper-regional solution of it. After that, we go back to the third step to re-partition the hyper-region into regions in a loop. Through this process, the partition becomes more reasonable and the solution is improved continuously. We can eventually obtain a solution with high quality.

4.2 Region Initialization

Owning to a direct intuition that the spatially close customers are more likely to be scheduled within the same route in an optimal solution, a division initialization based on the spatial locations is reasonable and will benefit to the convergence of the model's training. Therefore, we cluster customers according to their locations and divide the entire graph $G(V, E)$ into subgraphs G_i as the initialization. We adopt K-means in this step for its effectiveness [35]. To accommodate to the generator model, we set $K = \lceil \frac{N}{N_s} \rceil$, where N is the problem size and N_s is the small-scale problem size suitable with the generator model.

Besides, since the depot should be included in each local route, it is important to make full use of both customer locations and the depot location for clustering. Considering both relative distance and direction, we set the distance used for K-means as a linear combination of Euclidean distance d^E and polar distance d^P, which is calculated using the cosine of the included angle θ in the polar coordinate system, whose center is at the depot and the axis is a fixed line. The overall distance between customer i, j is $d_{i,j} = (1-\beta)d^E_{i,j} + \beta d^P_{i,j}$, where β is a hyperparameter. The initial partition in Fig 2 shows an example of how customers are divided only according to their polar distance, i.e., when $\beta = 1$. The detailed β selection experiments can be found in Appendix A.3.

4.3 Generating

For small-scale VRPs in merged hyper-regions, we implement VRP sub-solvers to generate local solutions, termed as the generator. The well developed VRP research on small and middle scales provide choices on multiple solutions. In this paper, we adopt both RL based and non-RL based sub-solvers as follows,

- Following [20], we construct the Attention Model (AM) as the RL based end-to-end sub-solver. The encoder-decoder structure selects a customer into the current partial tour. When a partial tour is constructed, it cannot be changed and the remaining problem is to find the following path from the last customer. The generator benefits from the strong context representation ability to guarantee the performance and a separate off-line inference stage to generate solutions within an extremely short period.
- As for non-RL based choices, we adopt LKH3 [19], which is one of the recent state-of-the-art solver for both CVRP and many other VRP variants. Besides, we also adopt HGS [33], which is designed for CVRP only.

It is worth noting that the choice of generator is quite flexible. One can easily replace it with other efficient learning models for small-scale CVRP, and the overall framework remains unchanged. RBG has the potential to accept new advanced small-scale VRP solvers as its generator.

4.4 Rewriting

The rewriter agent conducts a **partitioning-selecting-merging** process to update region divisions. To be detailed, we first partition each hyper-region based on the inside routes generated by the generator into two smaller regions, then consistently employ the neural region representations to select region-pairs and merge them into hyper-regions.

We denote the k-th hyper-region by G_k, and the k-th region by G'_k. The depot v_0 is duplicated and obtained by every G_k and G'_k, which is a condition that feasible solutions exist in every local regions. The local CVRP solution in hyper-region G_k is denoted by $\pi_k = \{\tau_{k,r}\}, r = 1, ..., M_k$, where $\tau_{k,r}$ is the r-th route in the solution π_k and M_k is the number of routes in π_k. Similarly, the solution in region G'_k is denoted by $\pi'_k = \{\tau'_{k,r}\}, r = 1, ..., M'_k$. We use $|G'|$ or $|G|$ to represent the number of customers in region G' or hyper-region G.

Partitioning. We partition each hyper-region G_k into two smaller-size region G'_i and G'_j with similar number of customers, while at the same time keeping the generated routes $\tau_{k,r}, r = 1, ..., M_k$ in solution π_k unchanged. In other words, the partition is done upon the route level.

$$\begin{aligned} G'_i \cup G'_j &= G_k, & \pi'_i \cup \pi'_j &= \pi_k, \\ G'_i \cap G'_j &= \{v_0\}, & \pi'_i \cap \pi'_j &= \emptyset. \end{aligned} \tag{6}$$

Note that the similar numbers in G'_i and G'_j helps to maintain the regional-scale consistency for further process. Due to the same intuition addressed in the initialization that an optimal solution is more likely to assign close customers into the same route, we calculate the spatial center of all customers within one route as the route's representations and use principal component analysis (PCA) to reduce the representations to 1 dimension. Then we divide all routes by sides into two new regions G'_i and G'_j with similar amount of customers.

Selecting. After partitioning a hyper-region G_k into two regions G'_i, G'_j, we further generate regional representations via a neural network. We first represent each route consisting of a sequence of nodes as $\tau'_{i,r} = (v_0, \tau'_{i,r,1}, \tau'_{i,r,2}, ..., \tau'_{i,r,n_{i,r}}, v_0), r = 1, ..., M'_i$ inside region G'_i. The original 3-dimensional representation $\tau'_{i,r}$, including the coordinate and demand, is further processed via a d_h-dimensional MLP layer and an LSTM [14] layer to capture the sequential features. Further more, we take the mean value of the outputs from LSTM of each route in the region and process it using a fully connected network to generate the d_h-dimensional feature

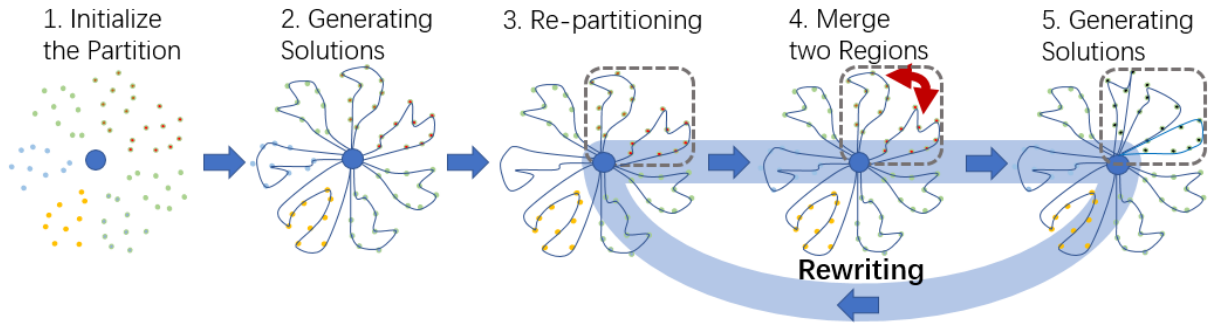


Figure 2: The working flow of the hierarchical RBG framework with five steps. The generator is responsible for a solution initialization in step 2. The iteration including both agents is shown in step 3, 4 and 5.

h_i of G'_i,

$$h_i = W_1^\tau [\frac{1}{M_i} \sum_{r=1}^{M_i} \text{LSTM}(W_0^\tau \tau'_{i,r} b_0^\tau)] + b_1^\tau, \tag{7}$$

where W_1^τ and b^τ are shared weights and bias respectively, M_i is the number of routes in region G'_i and $h_i \in \mathbb{R}^{d_h}$ is the representation of each region G'_i for further process. All regions share the same region encoder and its parameters.

With the well represented regional features, we thus select region-pairs for further merging. Fixing G'_i as one candidate region, we now select another region for further merging. We compute the selection probability $p_{i,j}$ of G'_i and G'_j as

$$p_{i,j} = \text{softmax}_{j \in U_i}(h_i^T h_j), \tag{8}$$

where U_i is the set of the nearest regions to G'_i, and $p_{i,j} = 0, \forall j \notin U_i$. We set $|U_i| = K = 5, 8, 9$ with $N = 500, 1000, 2000$ respectively. Such a reduction on selection space could help the HRL framework to converge much faster. Then we use softmax to select the target region G'_j for the following merging process.

Merging. Finally, we obtain a set of region-pairs (G'_i, G'_j), and we merge each region-pair together into a new hyper-region $G_{merge} = G'_i \cup G'_j$, in which we will regenerate solutions by the generator in the next iteration step.

Now an iteration of Rewriting-by-Generating is completed. We perform a fixed number of such iterations at the training and evaluation stages respectively. At the t-th iteration step, which is also termed as the rollout step or rewriting step, we regenerate the corresponding hyper-regional CVRP solution π^t_{merge} over the merged hyper-region G_{merge} using the elementary CVRP generator. Since π^t_{merge} considers the customer information more globally, it is comprehensible that the newly generated solution π^t_{merge} is more likely to obtain better quality than $\pi'^t_i + \pi'^t_j$. If the new solution is better than the previous one, we accept the updating on the overall solution π^t of all customers as follows,

$$\pi^{t+1} = \pi^t - \pi'^t_i - \pi'^t_j + \pi^t_{merge}. \tag{9}$$

To summarize, the rewriter takes a partitioning-selecting-merging process to update region divisions. Such a step is called a rewriting step or a rollout step. The selecting part of the rewriter is optimized with the performance improvement of the new solutions to the previous one.

4.5 Optimization via REINFORCE

In the above-mentioned steps, the partition takes place within the route-level, which does not change the total distance cost. This means the total cost is only influenced by merging and regenerating. Hence we define the reward function for one region-pair (G'_i, G'_j) as follows,

$$r^t = C(\pi'^t_i) + C(\pi'^t_j) - C(\pi^t_{merge}), \tag{10}$$

where $C(\pi)$ is the total tour length of solution π. It describes how the solution is improved in each iteration. The gradient updating of rewriter's parameter θ in a batch B at rollout step t is as follows,

$$\nabla_\theta L_r(\theta) = \frac{1}{B} \sum_{b=1}^{B} (r_b^t - b_t) \nabla_\theta \log \mathbb{P}_\theta(G_{b,merge}) \tag{11}$$

where b_t is the baseline of REINFORCE method [36] to reduce the variance of the gradient propagation. It is updated as the running average of rewards. Meanwhile, to guarantee the iteration effectiveness, if the newly generated solution is worse than the previous one, we reject the updated partition and solutions, and forward to the next rewriting step.

5 PERFORMANCE EVALUATION

In this section, we conduct extensive experiments and answer the following three research questions:

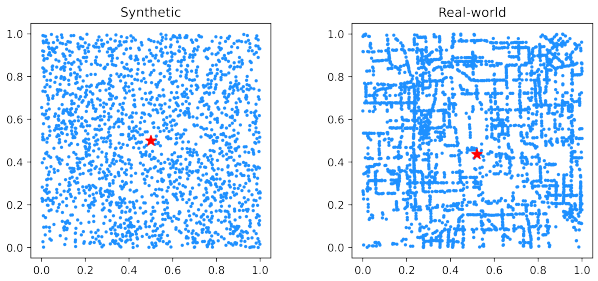
- **RQ1:** How does our proposed RBG framework perform compared to other state-of-the-art methods on large-scale VRPs?
- **RQ2:** How well can RBG generalize to other variant routing problems with more practical constraints in application?
- **RQ3:** How is the robustness of RBG when facing different customer distributions in practice?

Besides offline performance analysis, we also deploy RBG to a deployed logistic platform online to generate routing solutions, which is demonstrated further in Section 6.

5.1 Experiment Settings

5.1.1 Datasets. We evaluate the performance of RBG on two datasets, including a synthetic one and a real-world one as follows.

- **Synthetic Dataset.** We follow the same data generation method as the CVRP evaluation settings from previous works [9, 20, 26] for consistency and fair comparison. The location of each customer (x_i, y_i) is sampled uniformly from a unit square. The demand d_i of each customer is sampled from the discrete set $\{1, 2, ..., 9\}$, and the capacity of each vehicle is set 50. Unless otherwise stated in the robustness studies, we set the depot at the central of the area for simplicity. The traveling distance between two customers or the depot are calculated directly using the inner Euclidean distance. The test dataset contains 1000 instances for each problem size and is generated using different random seeds from the training and validation dataset.
- **Real-world Dataset** We collect the real-world dataset from an online logistic system within a week. The customer locations are randomly selected from the historical order records within a week in Beijing in June, 2020. The discrete demand of each customer node is also uniformly chosen from $\{1, ..., 9\}$, while the capacity is set as 50. The distances in-between are retrieved from GIS platform, and both location coordinates and distances are normalized into a unit square. The dataset distribution of both datasets are presented in Figure 3.



(a) Synthetic (b) Real-world

Figure 3: Data distributions of both synthetic and real-world datasets.

Training and Evaluation Details. For both datasets, we consider three different problem scales for large-scale CVRP with customer amount $N = 500, 1000, 2000$ respectively. We run 10 rollout steps for rewriting during each epoch and each model is trained for 50 epochs, and each epoch contains 100 problem instances. During evaluation, we run 100 rollout steps for rewriting. More details on training and evaluation for reproducibility are included in Appendix A.1.

5.1.2 Methods and Baselines. We first present a background performance for both datasets:

- **Random:** Routing solutions are generated randomly.

Then We compare the RBG framework with the four widely recognized heuristics methods:

- **Ant Colony Optimization(ACO)** [13] is a famous heuristic to solve the the basic VRP and its variants. A number of ant colonies are established to model and optimize the objective functions.
- **LKH3** [19] is one of the recent state of the art CVRP solver which adopts the classical Lin-Kernighan heuristic to obtain high-quality solutions [24]. We record both pre-computation time and iteration time.
- **HGS** [33] is a CVRP-specific solution designed especially for large scale cases.
- **OR Tools** [18], Google's vehicle routing problem solver that utilizes a set of metaheuristics, which is open online[1].

We also compare the most up-to-date RL-based approaches:

- **ReWriter** [9] proposes to keep rewriting current routing solutions to improve the routing quality.
- **AM** [20] utilize the transformer structure in both encoder and decoder. Since the origin algorithm can only solve instances with Euclidean distances only, we adapt it to the real-world data by projecting the real distance matrix into initial node embeddings. The model is trained on 100 instances and directly evaluated on large scales.
- **L2I** [25] establishes a learning-to-iterate framework by construction an operation pool, and train an agent to select an operator each time to conduct routing improvement.
- **L2D** [23] as a homochronous work as our RBG framework, iteratively selects subproblems out of all routes and delegates their improvement by a subsolver.

To evaluate the effectiveness and flexity of RBG, three RBG extended frameworks, RBG-AM, RBG-LKH, RBG-HGS are implemented with AM, LKH, HGS respectively.

5.2 Performance Comparison (RQ1)

Table 1: Overall performance comparison on the synthetic dataset. The best result is in bold and the second is underlined. The object *Obj.* is the average total travel distance. *Time* is the average time to solve a single instance.

	N = 500		N = 1000		N = 2000	
	Obj.	Time	Obj.	Time	Obj.	Time
Random	273.49	-	546.12	-	1091.35	-
ACO [13]	61.55	20min	112.28	50min	207.56	2h
LKH3 [19]	49.55	1s+5min	92.98	6s+15min	178.23	1min+40min
HGS [33]	<u>49.18</u>	15s	92.82	61s	177.20	180s
OR Tools [18]	54.72	20s	100.77	80s	186.95	5min
ReWriter [9]	60.67	33s	108.82	37s	198.76	8min
L2I [25]	52.58	74s	99.21	3min	191.34	17min
AM [20]	55.48	-	106.60	-	218.68	-
L2D [23]	49.54	60s	92.78	144s	176.12	356s
RBG-AM	51.23	**7s**	95.96	**15s**	181.43	**30s**
RBG-LKH	49.48	70s	<u>92.77</u>	159s	<u>175.79</u>	5min
RBG-HGS	**49.13**	15s	**91.74**	59s	174.50	90s

The comparison results on both datasets are shown in Table 1 and Table 2 respectively. RBG engaged with HGS as the generator shows consistent advantage to all other baselines in all experiments, and outperforms the best baseline by 1.66% with $N = 2000$ in the real-world dataset. While RBG-LKH is only slightly outperformed by LKH3 in two datasets with $N = 500$ by 0.2% and 0.6%, and still outperforms others in larger scales. It is interesting to find that the performance gaps of RBG-HGS and RBG-LKH compared to LKH3

[1]https://developers.google.com/optimization/routing/vrp.html

Table 2: Overall performance comparison on the real-world dataset. The best result is in bold and the second is underlined. The object *Obj.* is the average total travel distance. *Time* is the average time to solve a single instance.

	N = 500		N = 1000		N = 2000	
	Obj.	Time	Obj.	Time	Obj.	Time
Random	272.31	-	546.02	-	1095.62	-
ACO [13]	60.26	20min	109.80	50min	203.71	2h
LKH3 [19]	49.76	1s+2min	93.04	14s+20min	179.52	1.7min+1h
HGS [33]	<u>49.42</u>	15s	93.53	30s	178.81	120s
OR Tools [18]	55.4	5min	100.03	10min	187.01	30min
AM [20]	54.96	-	105.00	-	219.53	-
L2D [23]	49.62	83s	92.80	221s	177.83	359s
RBG-AM	51.22	**7s**	95.42	**15s**	180.99	**30s**
RBG-LKH	49.50	70s	<u>92.62</u>	3min	<u>176.47</u>	6min
RBG-HGS	**49.37**	**12s**	**92.33**	15s	**174.87**	88s

and HGS, the state-of-the-art heuristic algorithms, keep increasing when the size of customers increases. This demonstrates that our RBG framework not only generates high-quality solutions when the problem is at a large scale but also has a growing advantage to other methods when the problem scale grows. This is of great practical value since maintaining solution quality is critical to many realistic industrial applications in which a vast number of customers may occur frequently [2]. We also notice that the RBG framework outperforms the L2D structure consistently in all settings. This is because that selecting subproblems for the generator via the re-partitioning and merging cycles help to maintain the stability of region construction, compared to the complete start-over manner in L2D.

Apart from the solution quality, RBG also has remarkable inference efficiency. RBG-AM only takes 30s averagely to obtain the solutions to the instances with $N = 2000$. Compared to LKH3, one of the best heuristic methods that has a close performance to RBG, our framework works about 80× times faster. Even if RBG is eqiuped with relatively slower heuristic solvers, RBG-HGS and RBG-LKH still performs much more efficiently than the pure generator inference. A plausible reason is that solving partial instances by the generators locally prevents the framework from facing the entire global searching space . Consequently, the ability of fast-responding to new instances makes our proposed RBG framework adaptive to situations where instances require real-time responses, which is of great importance. For example, demands in on-demand delivery service change continuously due to new demand arrivals and order cancellations. Generating new solutions to new order distributions within time limits for fast-response is vital to a robust system design.

5.3 Generalization to Practical Routing Problems (RQ2)

In real-world applications, routing problems are usually with practical constraints. A practical framework should be capable of easily generating to other variant problems. Thus we analyze the how well can RBG generalize to two typical variants, CVRP with Time Windows (CVRPTW) and CVRP with Mixed Delivery and Pickup (CVRPMDP). We introduce the two problems and the datasets used as follows:

Table 3: The performance comparison with baselines on generalized practical routing problems. The best result in each column is in bold.

		N = 500		N = 1000		N = 2000	
Problem	Model	Obj.	Time	Obj.	Time	Obj.	Time
CVRPTW	LKH3	**62.84**	1.5min+90s	113.89	15min+5min	212.15	1.4h+20min
	L2D	63.74	1min	115.20	123s	213.62	5min
	RBG-LKH	63.48	30s	**113.04**	121s	**210.85**	5min
CVRPMDP	LKH3	55.75	45s+2min	103.32	7min+10min	196.72	44min+42min
	L2D	55.65	54s	102.13	130s	193.00	7min
	RBG-LKH	**55.10**	60s	**101.97**	121s	**191.79**	7min

- **CVRPTW** adds additional constraints on the arrival time of vehicles at each customer, which is restricted within a given time window $[e_i, l_i]$, where e_i and l_i are the frontiers and postiers of the time windows respectively. All vehicles are required to visit the customers within the time windows. Following the data generation procedure in Solomon and Li et al. [23, 30], the time windows are set based on the synthetic dataset by 1) sampling the time window center c_i $\mathcal{U}([e_0 + t_{0,i}, l_0 - t_{0,i} - s_i])$, where $t_{0,i}$ is the travel time equaling the distance in-between 2) sampling the time window hald-width h_i uniformly from $[0.1, 1]$ and 3) setting the entire time window as $[max(e_0, c_i - h_i), min(l_0, c_i + h_i)]$.
- **CVRPMDP** fomulates when there exists heterogeneous parcel parcel types, where some require delivering and some require picking up. The vehicles are required to accomplish both pickup and delivery tasks along the routes. Following the data generation procedure in Salhi et al. [28], we select half customers with pickup demands and the others with delivery demands based on the synthetic dataset, both sampled uniformly from $[1, ..., 9]$. The capacity of each vehicle is set as 25.

Since LKH3 [19] is the only stable generator which is capable to solve multiple VRP variants, we only compare RBG-LKH with other available baselines, including LKH3 itself and L2D.

The overall performance results are shown in Table 3.Except that RBG-LKH is slightly outperformed by LKH3 in CVRPTW with $N = 500$, the RBG based framework outperforms both baselines on both problem variants. Compared to pure LKH3, RBG-LKH obtains the optimization advantage by 0.62% and 2.5% on two problems respectively. Besides, RBG still shows its consistent efficiency in generating solutions up to 6x times faster. We also notice that the original LKH3 spends a great amount of time in pre-computation when facing complicated routing scenarios in large scales, while the region decomposition mechanism of RBG greatly saves such time. Compared to L2D, RBG benefits from its relatively stable region decomposition, so that the framework can improve solution quality faster.

While most RL based framework are hard to be adapted to numerous practical routing scenarios, RBG is flexible enough to be easily generalized by enhancing with corresponding generators. Such a generalization ability is of great importance in practical system deployment.

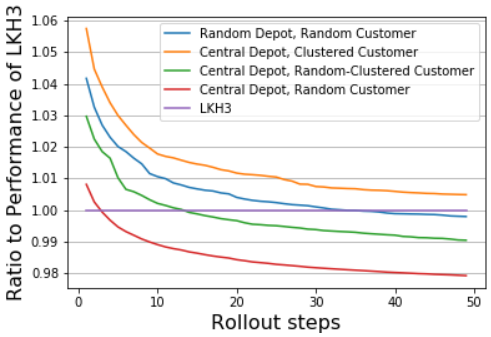


Figure 4: Performance on different data distributions.

5.4 Analysis of Robustness to Data Distribution (RQ3)

Practical online logistic systems takes customer orders as new data continuously. To further evaluate the robustness of RBG to different data distributions, we conduct experiments by generating four synthetic datasets with different distribution protocols following the widely accepted CVRP instance benchmark by Uchoa et al. [32]: 1) **Random depot, random cluster**, where the depot and the customers are all located randomly. 2) **Central depot, clustered customer**, where customers distribute in several clusters, 3) **Central depot, random-clustered customer**, where half of the customers are randomly located in a uniform distribution, and the other half gather at several clusters, 4) **Central depot, random customer**, which is the same with our training setting, i.e., the depot is located at the center and customers are located randomly in a uniform distribution. In all scenarios, the RBG model for evaluation is trained on the same central depot and random customer data, and then evaluated on different data distributions. We compare the performance in each scenario with LKH3, the baseline with the best performance. For clarity, the performance of RBG is normalized by the corresponding LKH3 traveling cost in Fig 4. The detailed data distribution is shown in Appendix A.5.

We find that our proposed RBG performs well across different data distributions and the improvement trends along rollout steps of the traveling cost are similar. Specifically, RBG outperforms LKH3 at about the 5-th, 10-th and 50-th step in the last three groups. Clustered customer is the only case where RBG is outperformed by LKH3. A plausible reason is that the clustering of the customer distributions make it easier for the LKH3 to find the close customers within the same cluster and assign them to the same route,and thus the advantage of dividing customers into reasonable regions is weakened. However, RBG still obtains high-quality solutions with performance loss of no more than 0.5%. Detailed visualization and statistics are presented in Appendix A.5.

6 ONLINE SYSTEM DEPLOYMENT AND EVALUATION

We integrate RBG into an online logistic system deployed in Guangdong, China. The platform collects delivery demands one day in advance, and provides routing strategies to a fleet to transport the parcels. Base on the given problem instances, the built-in algorithm computes the near-optimal routing solutions for the vehicle fleet

Table 4: The performance evaluation of several methods one day's logistic task. Results of the deployed algorithm, RBG-HGS, generated online is in bold, while others are evaluated offline later with the same data.

Model	500 by 500		700 by 500		700 by 1000	
	Obj.	Time	Obj.	Time	Obj.	Time
LKH3	32.12	0.6s+5min	39.71	2s+15min	39.71	2s+15min
HGS	31.85	60s	39.48	90s	39.48	90s
L2D	33.11	36s	40.57	36s	40.61	37s
RBG-LKH	32.25	17s	39.80	98s	39.80	99s
RBG-HGS	**31.84**	14s	**39.44**	**21s**	**39.46**	**21s**

in advance. Under circumstances where new demands arrive when the fleet already are already running and in service, the platform re-computes an updated routing solution considering the additional requests and send corresponding orders to the vehicles in real-time. Each vehicle in the fleet is equipped with an IoT device, so that the real-time coordinate, the remaining volume and the current working state of each vehicle can be captured by the centralized platform. A visualization of the platform is shown in Figure 5, where the real-time Gantt diagram describing working states of different vehicles is shown on the left, and the customers distributions are shown on the right. The green bars in the Gantt diagram indicates that the driver is not moving, either in service at the customer locations or in a break. While the blue bar indicates that the vehicle is running along the road guided by the GIS system.

To demonstrate the effectiveness of the proposed RBG framework, we compare the performances of different algorithms on two days' logistic tasks, with $N = 500$ and $N = 700$ each. All algorithms are pre-trained via the synthetic data. Since $N = 700$ is not a pre-established problem scale in the synthetic dataset, we evaluate the performance using model trained on both 500 and 1000 scales, termed as 700 by 500 and 700 by 1000. *LKH*3 and *HGS* are reported the same results in these two settings since they are not learning based approaches. Note that the only implemented algorithm is RBG-HGS, which generates solutions in the online environment. The entire comparison is finished by collecting the data and then evaluate other methods based on the data offline later.

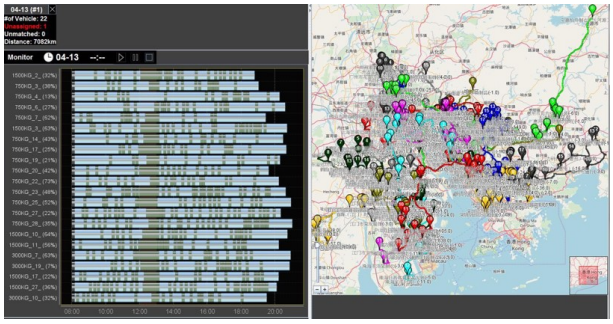


Figure 5: The interface of the online logistic platform deployed in Guangdong, China. A Gantt diagram describing the routing process of each vehicle is shown on the left, while the entire customer distribution is shown on the right.

The detailed comparison results are shown in Table 4. RBG-HGS shows its consistent performance compared to other selected

baselines. In the $N = 500$ task, the previous SOTA baseline HGS [33] takes 60s to obtain close solution quality as RBG-HGS, who only takes 14s to finish the searching progress till near convergence. It is also worthy to note that RBG-HGS performs steadily on 700 by 500 and 700 by 1000, which indicates that the RBG framework is also flexible in generalizing across different problem scales. Such a generalization feature is extremely important in online system implementation, since we cannot expect to train offline models on all problem scales. When solving online instances with a specific problem scale no model has ever been trained on, the platform can easily selects the model with close scales and could still maintain stable performances.

Our improvement is significant in practical logistic systems. An urgent problem is that when customer demands change rapidly, a platform using heuristics cannot generate a corresponding changed solution at once. Another situation is that if a qualified solution for a delivery task does not exist, we need to split customers into classes with different priorities and generate several solutions by steps. This may happen when the customer amount exceeds a maximum or some extremely distant customer occurs, while the vehicles are not enough for usage. Under both circumstances, real-time response to new problem inputs is of great significance. This is exactly where the RBG framework shows its advantage.

7 DISCUSSION AND CONCLUSION

In this paper, we propose the Rewriting-by-Generating framework for solving large-scale routing problems. The framework generates regional routing solutions within each independent regions via a generator, and rewrites its previous solution by merging and re-partitioning region pairs via an RL based rewriter. We demonstrate RBG generates high-quality solutions efficiently. We also showcase the robust ability for RBG to perform effectively in a variety of distributions that may be different from the training distribution.

8 ACKNOWLEDGEMENTS

This work was supported in part by the National Key Research and Development Program of China under grant 2020AAA0106000, the National Natural Science Foundation of China under U1936217, 61971267, 61972223.

REFERENCES

[1] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. 1997. An efficient k-means clustering algorithm. (1997).
[2] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. 2019. Efficiently solving very large-scale routing problems. *Computers & Operations Research* 107 (2019), 32–42.
[3] Barrie M Baker and MA Ayechew. 2003. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research* 30, 5 (2003), 787–800.
[4] Roberto Baldacci, Enrico Bartolini, Aristide Mingozzi, and Roberto Roberti. 2010. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science* 7, 3 (2010), 229–268.
[5] Roberto Battiti and Giampietro Tecchiolli. 1994. The reactive tabu search. *ORSA journal on computing* 6, 2 (1994), 126–140.
[6] John E Bell and Patrick R McMullen. 2004. Ant colony optimization techniques for the vehicle routing problem. *Advanced engineering informatics* 18, 1 (2004), 41–48.
[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural Combinatorial Optimization with Reinforcement Learning. (2017).
[8] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. 2016. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering* 99 (2016), 300–313.
[9] Xinyun Chen and Yuandong Tian. 2019. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*. 6278–6289.
[10] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. (2017), 6351–6361.
[11] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. 2020. Reinforcement learning with combinatorial actions: An application to vehicle routing. *arXiv preprint arXiv:2010.12001* (2020).
[12] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. 2006. Ant colony optimization. *IEEE computational intelligence magazine* 1, 4 (2006), 28–39.
[13] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. 1999. MACS-VRPTW: A Multiple Colony System For Vehicle Routing Problems With Time Windows. In *New Ideas in Optimization*. McGraw-Hill, 63–76.
[14] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).
[15] Fred Glover. 1990. Tabu search - Part I. *INFORMS Journal on Computing* 2 (01 1990), 4–32.
[16] David E Goldberg. 1989. Genetic algorithms in search. *Optimization, and MachineLearning* (1989).
[17] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. 2008. *The vehicle routing problem: latest advances and new challenges*. Vol. 43. Springer Science & Business Media.
[18] Google. 2016. OR-Tools. https://developers.google.com/optimization/routing.
[19] Keld Helsgaun. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University* (2017).
[20] Wouter Kool, Herke Van Hoof, and Max Welling. 2019. Attention, Learn to Solve Routing Problems! (2019).
[21] K Krishna and M Narasimha Murty. 1999. Genetic K-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29, 3 (1999), 433–439.
[22] Gilbert Laporte. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research* 59, 3 (1992), 345–358.
[23] Sirui Li, Zhongxia Yan, and Cathy Wu. 2021. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems* 34 (2021).
[24] Shen Lin and Brian W Kernighan. 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations research* 21, 2 (1973), 498–516.
[25] Hao Lu, Xingwen Zhang, and Shuang Yang. 2020. A Learning-based Iterative Method for Solving Vehicle Routing Problems. In *International Conference on Learning Representations*.
[26] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takac. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. (2018), 9861–9871.
[27] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. 2020. A generic exact solver for vehicle routing and related problems. *Mathematical Programming* 183, 1 (2020), 483–523.
[28] Saïd Salhi and Gábor Nagy. 1999. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the operational Research Society* 50, 10 (1999), 1034–1042.
[29] Gerhard Schrimpf, Johannes Schneider, Hermann Stamm-Wilbrandt, and Gunter Dueck. 2000. Record Breaking Optimization Results Using the Ruin and Recreate Principle. *J. Comput. Phys.* 159, 2 (2000), 139 – 171. https://doi.org/10.1006/jcph.1999.6413
[30] Marius M Solomon. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research* 35, 2 (1987), 254–265.
[31] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
[32] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. 2017. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research* 257, 3 (2017), 845–858.
[33] Thibaut Vidal. 2022. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* Neighborhood. *Computers & Operations Research* 140 (2022), 105643.
[34] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*. 2692–2700.
[35] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. 2001. Constrained k-means clustering with background knowledge. In *Icml*, Vol. 1. 577–584.
[36] Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (01 May 1992), 229–256. https://doi.org/10.1007/BF00992696
[37] Huizhen Zhang, Qinwan Zhang, Liang Ma, Ziying Zhang, and Yun Liu. 2019. A hybrid ant colony optimization algorithm for a multi-objective vehicle routing problem with flexible time windows. *Information Sciences* 490 (2019), 166–190.

A APPENDIX

A.1 Additional Training and Evaluation Details

Training. During the training process, we use a learning rate of 0.001 to train the rewriter optimized by SGD. The dimension of route embedding d_h is set as 100. For each instance, we run K-means for 10 iterations in the initialization step and set $\beta = 0.09$. We run 10 rollout steps for rewriting during training and randomly rotate the positions of all customers along the depot at each step to make training data in a near-i.i.d. distribution to prevent overfitting. Each model is trained for 50 epochs, and each epoch contains 100 samples. After each epoch, we evaluate the model on the validation dataset with 20 samples and update the current model if the performance improves. The updating parameter α for the baseline of REINFORCE is set as $\alpha = 0.99$. Our method is implemented in Python via Pytorch framework, and the experiments are run on one Nvidia 2080Ti GPU.

Evaluation. During evaluation, we maintain the same generator setting as in the training process. As for the rewriting process, we run 100 rollout steps for each problem instance. All evaluation results are reported based on the average of performance of 100 instances, in terms of the objective and solving time. Note that LKH spends significantly long time during pre-processing before reporting the first result on large-scale VRPs, so we report such a time individually.

For both datasets, we consider three different problem scales for large-scale CVRP with customer amount $N = 500, 1000, 2000$ respectively.

A.2 Analysis of Rewriting Strategy

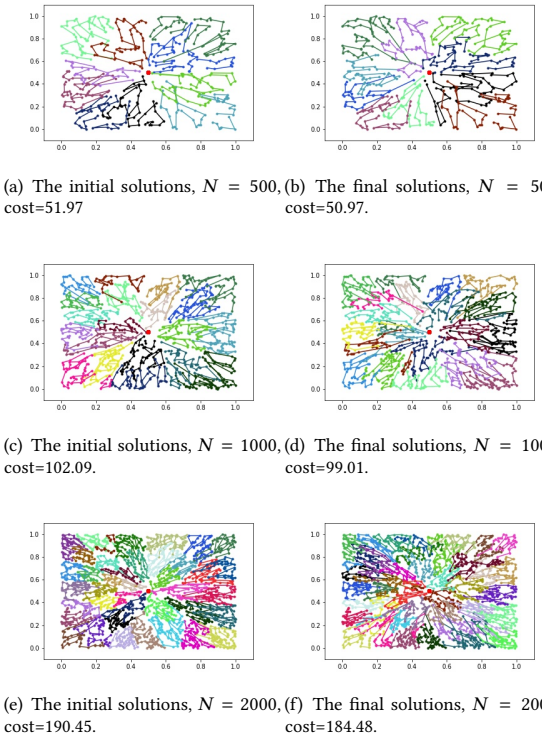
Instead of analyzing only final comparison with other baselines, we also look into the detailed iteration steps to analyze how RBG improves its solutions iteratively.

We analyze the extent and the frequency of partition along with the performance improvement. Figure 7(a) shows the reallocation ratio R by rollout steps, which is the ratio of customers reallocated after a rewriting operation. It is calculated as follows,

$$R = \sum_{(G'_i, G'_j)-\text{pair}} \frac{min(|G'_i - G'_j \cap G'_{i'}|, |G'_i - G'_i \cap G'_{j'}|)}{|G|} \tag{12}$$

where a region pair (G'_i, G'_j) is merged into a hyper-region and then re-partitioned into a new region pair $(G'_{i'}, G'_{j'})$, and G is the entire graph. Figure 7(b) shows the rate of repartition being accepted. Figure 7(c) shows the normalized traveling cost ratio to the minimal cost along with rollout steps.

We find that the reallocation ratio and the partition update rate decreases significantly as long as the traveling cost decreases. The similar decreasing trends among them demonstrate that the improvement of the solution quality is highly related to the extent and frequency of rewriting. The rewriter finally tends to stop its rewriting operation when the solutions are close to optimal. In conclusion, the rewriter shows its effectiveness in improving the solution quality during the instance inference.



(a) The initial solutions, $N = 500$, cost=51.97

(b) The final solutions, $N = 500$, cost=50.97.

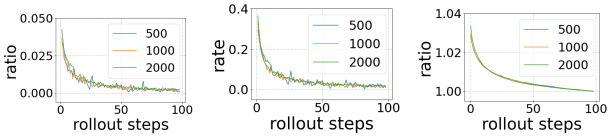
(c) The initial solutions, $N = 1000$, cost=102.09.

(d) The final solutions, $N = 1000$, cost=99.01.

(e) The initial solutions, $N = 2000$, cost=190.45.

(f) The final solutions, $N = 2000$, cost=184.48.

Figure 6: Visualization on initial and final solutions of different scales. The big red point at the center represents the depot, and each blue point represents a customer. Routes with the same color are in the same region. For clarity in visualization, we omit the line segment from depot to the first customer and from the final customer back to depot for each route.

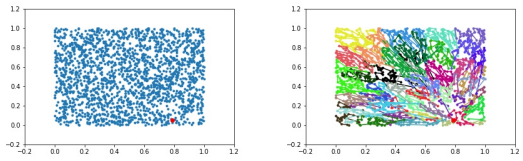


(a) Reallocation (b) Update Rate (c) Ratio

Figure 7: Statistics of the rewriter at different scales as the rewriting step increases. (a) shows the change of reallocation ratio R defined by equation 12. (b) shows the rate of repartition being accepted. (c) shows the ratio of travelling cost to the minimal cost available.

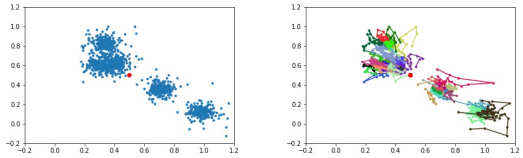
A.3 Analysis on different initialization strategies

Due to the intuition that the closely-distributed customers are more likely to appear in the same route in an optimal solution, we cluster the customers according to their locations as discussed in Section 3. However, the detailed spatial feature used for clustering may



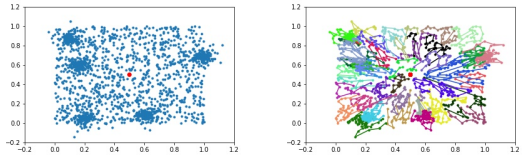
(a) $N = 2000$ instance of random depot&random customer data.

(b) Solution, cost=274.17.



(c) $N = 2000$ instance of central depot&clustered customer data.

(d) Solution, cost=146.35.



(e) $N = 2000$ instance of central depot&random-clustered customer data.

(f) Solution, cost=199.71.

Figure 9: Visualization on graphs and solutions of different data distributions. The big red point represents the depot, and each blue point represents a customer. Routes with the same color are in the same region. For clarity in visualization, we omit the line segment from depot to the first customer and from the final customer back to depot for each route.

vary. Hence we analyze two different ways to measure the related distance between two customers i and j: 1) **Euclidean distance**, $d^E_{i,j}$ and 2) the **polar distance**, $d^P_{i,j}$, which is calculated using the included angle θ in the polar coordinate system, whose center is at the depot and the axis is a fixed line.

To measure the influence of the two partition features, we use a combination of them, $(1-\beta)d^E_{i,j} + \beta d^P_{i,j}$, to represent the distance between customer i and j, where β is a hyperparameter. The initialization is obtained via the classical K-means algorithm[1, 21] and the results are shown in Fig 8. The traveling costs are normalized by the minimal one. We find that the generated solutions has the best quality when $\gamma = \frac{\beta}{1-\beta} = 0.1$, i.e., $\beta = 0.09$. Euclidean distance is a straightforward feature to measure the closeness of customers, but the division that relies solely on it may generate clusters that are far away from the depot. The corresponding route may suffer from a great distance cost to travel from the depot to the cluster, and then back to the depot. While the polar distance can prevent this shortage. An appropriate combination of them can benefit the RBG framework to obtain higher performance. It is also remarkable to

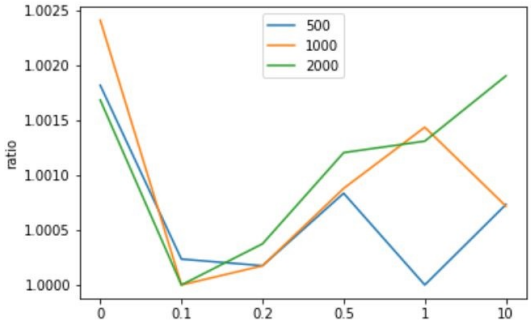


Figure 8: Ablation study on distance factor($\gamma = \frac{\beta}{1-\beta}$) for clustering. x-axis is different γs, and y-axis is the traveling costs normalized by the minimal one.

point out that the spatial feature combination has a low fluctuation range. The worst performance in every customer scales is no worse than 0.25% than the optimal one. This shows the great robustness of RBG to different initialization strategies.

A.4 Visualization of Inference

We present the visualization of the initial and final solutions in all three customer scales, as shown in Fig 6. It is worthy to mention that due to the initialization based on spatial features, the initial routes appear to be more gathering in space. Different routes can usually be separated by a clear boundary between them. However, in the final solutions which are rewritten and regenerated for steps by RBG, the routes tend to have more complicated intersections in space. This is because other factors, including the customer demands and the capacity are further considered by RBG and more reasonable divisions and routes are updated.

A.5 Visualization of Solutions with Different Data Distribution

We also present the visualization results on different instances with different data distributions, shown in Figure 9.