Learning to Delegate for Large-scale Vehicle Routing

Anonymous Author(s)
Affiliation
Address
email

Abstract

Vehicle routing problems (VRPs) are a class of combinatorial problems with wide practical applications. While previous heuristic or learning-based works achieve decent solutions on small problem instances of up to 100 customers, their performance does not scale to large problems. This article presents a novel learning-augmented local search algorithm to solve large-scale VRP. The method iteratively improves the solution by identifying appropriate subproblems and *delegating* their improvement to a black box subsolver. At each step, we leverage spatial locality to consider only a linear number of subproblems, rather than exponential. We frame subproblem selection as a regression problem and train a Transformer on a generated training set of problem instances. We show that our method achieves state-of-the-art performance, with a speed-up of up to 15 times over strong baselines, on VRPs with sizes ranging from 500 to 3000.

1 Introduction

Vehicle routing problem (VRP) has enjoyed ample applications in logistics and ride-hailing services [17] around the world for decades. While determining the optimal solution to a VRP is NP-hard [19], there have been numerous attempts to solve VRPs both exactly and approximately: provable algorithms have been designed for specific problem instances up to size 130 [22], and powerful heuristic solvers such as LKH-3 [10] find good solutions in practice for problems of size more than 1000. However, heuristics methods often suffer from inflexibility due to extensive hand-crafting and the heavy computational burden from lengthy iterative procedures, as in the case of LKH-3. For example, LKH-3 takes more than an hour to solve a size 2000 CVRP instance (Table 1), which is impractical for applications such as real-time ride-hailing services.

More recently, machine learning methods inspired by the Pointer Network [36] provide an alternative to traditional solvers: the learning-based methods are able to greatly reduce computation time while maintaining decent solution quality on small problem instances (size less than 100), by training the network on a diverse set of problem distributions either via supervised [36] or reinforcement learning [23, 14]. However, these methods remain difficult to scale up, and only a few report results on problems of size more than 200.

Our work aims to address scalability by learning to identify smaller subproblems which can be readily solved by existing methods. Our learned subproblem selector guides the problem-solving process to focus local improvement on promising subregions. While there exists a combinatorial number of subproblems that can be selected at each step, we leverage spatial locality commonly found in combinatorial optimization problems to restrict the selection space dimension to be linear in the problem size. Intuitively, objects far away from each other generally have very small influence on each other's solution, and should not be part of the same subproblem. The greatly reduced search space enables us to feasibly train an attention-based model to select subproblems.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Our framework combines the best of both worlds from learning and heuristics: our network identifies promising subproblems to improve upon, dramatically speeding up the solution time. Using LKH-3 as a subsolver for the subproblems, we achieve good solution quality without the high computational burden of running LKH-3 on large problem instances. In summary, our contributions are as follows:

- We propose a learning-based framework to solve large-scale VRPs by iteratively identifying and solving small subproblems.
- We use spatial locality to restrict the subproblem selection space, allowing us to feasibly train a subproblem selector for large problem instances.
- Extensive experiments show that our method achieves the state-of-the-art result in solution quality in various distributions of VRPs while using 7x to 15x less computation time than state-of-the-art solver LKH-3 [10].

2 Preliminary: Capacitated Vehicle Routing Problems (CVRP)

In CVRP, there is a depot node 0 and city nodes $\{1, ..., N\}$. Each city node has a demand d_i to fulfill. A vehicle with a certain capacity C starts at the depot and visits each city node i with demand d_i until the capacity C is exhausted, after which the vehicle returns to the depot to finish the route, and starts a new route from the depot again. The objective is to find a valid solution that minimizes the solution cost. Here we define the following terms:

- Route: a sequence of nodes, where the first and last node are depot 0, and the rest are city nodes. In a valid route, the sum of demands of the city nodes does not exceed C.
- Route cost: the sum of edge costs for the sequence of nodes. For an edge from node i to node j, the edge cost is the Euclidean distance between node i and j.
- Solution: a *feasible* solution consists of a set of valid routes visiting each city exactly once.
- Solution cost: the sum of route costs for all routes in the solution. An *optimal* solution is a feasible solution with the minimum solution cost.
- Subproblem: the depot node, a subset of city nodes, and corresponding demands.

3 Related Work

Classical methods. Heuristics for solving combinatorial optimization problems have been studied for decades. The most powerful methods, such as local search [1], genetic algorithms [27], and ant colony methods [8], involve iteratively improving the solution in a hand-designed neighborhood. For example, move, swap [38], and 2-opt [7] are well-known heuristics for the traveling salesman and vehicle routing problems. The state-of-the-art VRP solver LKH-3 [10] uses the Lin–Kernighan heuristic [20] as a backbone, which involves swapping pairs of sub-tours to create new tours.

For large problems, sophisticated *meta-heuristics* layer on top of low level heuristics, including Tabu Search with Adaptive Memory [31], guided local search [37], and Large Neighborhood Search [26]. The inspiration for our work derives from Partial OPtimization Metaheuristic Under Special Intensification Conditions (POPMUSIC), which iteratively optimizes on problem subparts, and has been used to solve problems as diverse as map labeling [18], berth allocation [16], and p-median clustering [30]. Despite the promise of the POPMUSIC framework for large-scale combinatorial optimization, the impact of certain design choices is not well understood, such as the subproblem selection ordering and size [33]. In particular, it is not clear how to meaningfully order the subproblems. One early work in this direction demonstrated that a last-in-first-out stack order performs better than random [3]. Our work provides a natural approach to order the subproblems based on expected improvement.

Deep learning methods. Recently, there has been a surge of interest in training deep neural networks (DNN) to solve combinatorial problems. As observed by Kwon et al. [15], most methods fall into one of the following two categories: 1) *construction methods* [23, 14], where an autoregressive model such as the Pointer Network [36] directly outputs a solution, and 2) *improvement methods* [5, 11], where the DNN iteratively performs local updates to the solution, resembling local search.

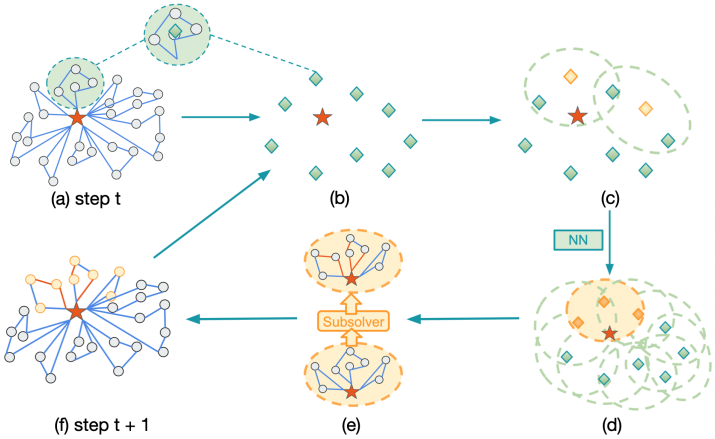

Figure 1: **Our iterative framework for VRP**. (a) At each time step, we start with a current solution X. Circles are city nodes, blue lines are routes, and the red star is the depot. (b) We aggregate each route by taking the centroid of all city nodes in the route. (c) For each route, we define a corresponding subproblem as the k-nearest neighbors of the route. Two such subproblems with $k = 3$ are shown. (d) Our subproblem selector selects a subproblem S (yellow). (e) We feed S into the subsolver to get a new subsolution X'_S. The red edges are updated by the subsolver. (f) We update X to new solution X' with X'_S, then repeat (b)-(f).

These methods are approaching the solution quality of LKH-3 [10] on small problem instances. For example, Kwon et al. [15] extends a constructive approach [14] to encourage more diverse output samples on $N \leq 100$ VRPs. Among improvement methods, Lu et al. [21] learn a meta-controller to select among a set of local search heuristics, marking the first learning-based approach to outperform LKH-3 on $N \leq 100$ VRPs. Despite these successes, learning-based approaches for large-scale VRPs are poorly understood. In particular, all of the aforementioned methods are trained using deep reinforcement learning; for large problems, trajectory collection becomes prohibitively expensive.

Scaling up learning methods. A few early works have just begun to investigate scaling of learned networks for NP-hard graph problems. For example, Ahn et al. [2] propose an effective iterative scheme called *learning-what-to-defer* for maximum independent set. At each stage, and for each node in the graph, the network either outputs the solution for the node, or defers the determination to later stages. Song et al. [29] proposes an imitation-learning-based pipeline for Integer Linear Programs (ILP), where at each stage they partition all variables into disjoint subsets, and use the Gurobi [9] ILP solver to solve the partitions. Due to differences in graph structure, our work presents a more natural scheme to handle VRP constraints and structures.

4 An Iterative Framework for CVRPs

While CVRPs are NP-hard and thus require worst-case exponential time in the problem size to solve optimally, typical CVRPs exhibit structure that practical solvers may exploit. We hypothesize that in such situations, the larger problem can be efficiently approximately solved as a sequence of smaller subproblems, which can be *delegated* to efficient subsolvers. To test this hypothesis, we propose a learning-based, iterative framework with two components: 1) a *subsolver* capable of solving a small problem instance (exactly or approximately), and 2) a *learned model* for identifying suitable subproblems within a larger problem to delegate to the subsolver.

We illustrate our iterative framework in Figure 1, which takes in a problem instance P with a feasible initial solution X_0. At each step, given the current solution X, we select a smaller subproblem $S \subset P$ with our learned model NN then apply the subsolver to solve S; we then update X with the new solution for S. To maintain feasibility after an update, we restrict S to be the set $\mathcal{S}$ of visited cities of a *subset of routes* from X. Since routes with cities in $P \setminus S$ remain valid routes, we obtain a new feasible solution $X' = X'_S \cup X_{P \setminus S}$, where X'_S is the subsolution for S and $X_{P \setminus S}$ consists of

Algorithm 1: Learning to Delegate

Input: Problem instance P, initialized solution X, subproblem selector NN, subsolver $Subsolver$, number of steps T, parameter k denoting the size of subproblems

for *Step t = 1: T* **do**
  $\mathcal{S}_{k,\text{local}} \leftarrow$ ConstructSubproblems(P, X, k)
  $S \leftarrow NN(\mathcal{S}_{k,\text{local}})$
  $X'_S \leftarrow$ Subsolver(S)
  $X \leftarrow X'_S \cup X_{P \setminus S}$
end for

unselected routes from X. Intuitively, a strong NN should identify a subproblem S such that the subsolver solution X'_S results in a large improvement in objective from X to X'.

4.1 The Restricted Subproblem Selection Space

As the number of routes in P is $R \in O(N)$, the cardinality of the selection space $\mathcal{S}$ is $O(2^R)$, which is exponential in the problem size N and difficult for learned models to consider. If we restrict each subproblem to cities from exactly k routes, there are still $\binom{R}{k} = O(R^k)$ subproblems to consider. Therefore we further restrict selection to subproblems with spatial locality. As shown in Figure 1(d), we only consider subproblems from $\mathcal{S}_{k,\text{local}}$, where each subproblem is centered around a particular route r and contains the k routes whose centroids have the smallest Euclidean distance to the centroid of r. In this way, we reduce the selection space to $|\mathcal{S}_{k,\text{local}}| = R = O(N)$ from $|\mathcal{S}| = O(2^N)$. In Algorithm 1, we refer to this restriction as ConstructSubproblems. Our restriction to a local selection space is motivated by the fact that many combinatorial optimizations problems have inherent spatial locality, i.e. problem entities are more strongly affected by nearby entities than faraway entities. Earlier heuristical methods such as POPMUSIC [32] leverage similar spatial locality.

5 Learning to Delegate

In this section, we discuss criteria for selecting subproblems, and how to train the subproblem selector.

Improvement as the criteria for subproblem selection. Given a selected subproblem S with a current solution X_S on the subproblem, we obtain from LKH-3 a new solution X'_S on the same subproblem (step (d) to (e) in Figure 1). We then define the *immediate improvement*

$$\delta(S) = c(X_S) - c(X'_S) \tag{1}$$

where $c(X)$ is the total cost of the solution X and $\delta(S)$ is the improvement in the solution cost. In this way, the sum of improvements along T steps is the total improvement in solution quality. As we empirically find that providing the previous subsolution X_S to the subsolver may trap the subsolver in a local optimum, we withhold X_S from the solver and thus may see non-positive improvement $\delta(S) \leq 0$, especially after many steps of subproblem selection.

Subproblem selection. The goal of our subproblem selector is to select the subproblem leading to the best *immediate* improvement. While our subproblem selection does not directly optimize for the total improvement, we observe that (1) our subsolver may perform many low-level operations internally, so high-level problem selection may still benefit greatly from maximizing the immediate improvement, and (2) numerous reinforcement learning approaches to VRP [39, 13] choose a small discount factor such as $\gamma = 0.25$ when optimizing a multi-step objective $\sum_{t=1}^{T} \gamma^t \delta_t$, as doing so encourage faster convergence. Moreover, our subproblem selector may instead be trained on multi-step search data to select subproblems offering the best multi-step improvement.

Ground-truth labels Our restricted selection space allows us to enumerate all possible subproblems at each step. In a typical large scale CVRP instance with $N = 2000$ cities, a solution consists of roughly $R = 200$ routes, so the size of the selection space is 200. We obtain the immediate

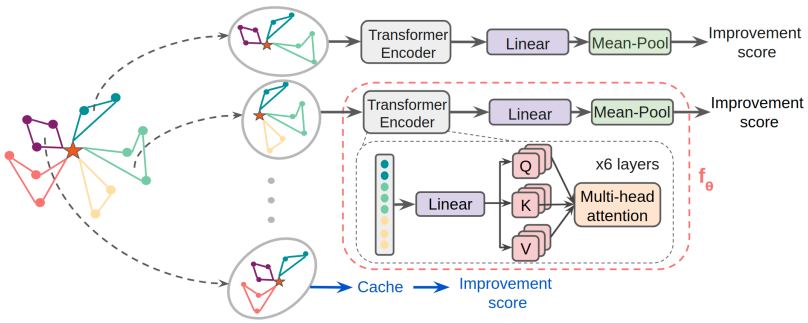

Figure 2: **Our transformer architecture**. At each step of our framework (Figure 1(c)), we featurize subproblem $S \in \mathcal{S}_{k,\text{local}}$ into a column vector of unordered cities. We apply the Transformer encoder with multiple multi-head attention layers and a final linear layer, before mean-pooling over the cities to generate the predictions $f_\theta(S)$, which we fit to the immediate improvement $\delta(S)$. When possible, we retrieve a previously predicted subproblem from a cache to minimize computation.

improvement $\delta(S)$ by running the subsolver on each subproblem $S \in \mathcal{S}_{k,\text{local}}$. Although our enumeration strategy is feasible for generating training data, it is much too slow to execute on a test CVRP instance. However, if our subproblem selector can identify the best immediate improvement at test time (that is, without running the subsolver on multiple subproblems), then we can combine the best of both worlds to obtain a fast and accurate selection strategy.

Selection strategies: regression vs classification Given a labeled dataset, we may learn a model to perform either regression or classification to identify the best subproblem. In the context of imitation learning with a greedy expert, the former learns a *value function* while the latter learns a *policy*. Due to space limitation, we focus discussion on regression and reserve the comparison with classification for Appendix A.1. The regression-based subproblem selector trains a regression model f_θ to predict $\delta(S)$ then simply computes $\arg\max_S f_\theta(S)$ to select the best subproblem.

Network architecture. We define f_θ based on the Transformer encoder architecture [35]. The input for each subproblem S is the unordered set of featurized cities in S. The features for each city consist of the demand d and the location of the city (x, y) relative to the depot. As we do not feed the existing subsolution X_S to the subsolver, the dense multi-head attention mechanism does not need to be modified to take the routes in X_S into account. The output of the Transformer encoder is fed into a linear layer then mean-pooled to obtain the scalar prediction $f_\theta(S)$.

Loss function. We define our loss function to be the Huber loss [12], which we empirically found to be more stable than mean squared error,

$$L(\theta; S) = \begin{cases} \frac{1}{2}(f_\theta(S) - \delta(S))^2, & \text{if } |f_\theta(S) - \delta(S)| \leq 1 \\ |f_\theta(S) - \delta(S)| - \frac{1}{2}, & \text{otherwise} \end{cases} \tag{2}$$

6 Experiments and Analysis

We illustrate the CVRP distributions considered in our work in Figure 3. We perform extensive experiments to evaluate our learning framework. We aim to answer the following questions:

1. **Uniform distribution**. How does our method compare with baselines on the uniform data distribution, in terms of solution time and quality?
2. **Clustered distributions**. How does our method perform on clusters of cities?
3. **Generalization**. Can our model generalize, such as to larger or real-world instances?
4. **Other VRP variants**. Can the method address more sophisticated VRPs, e.g., CVRP with Time Windows (CVRPTW) [28], VRP with Mixed Pickup and Delivery (VRPMPD) [25]?

Due to space limitation, we reserve additional ablation on the choice of subproblem size (parameterized by k) and additional analysis of asymptotic behavior for Appendix A.6.

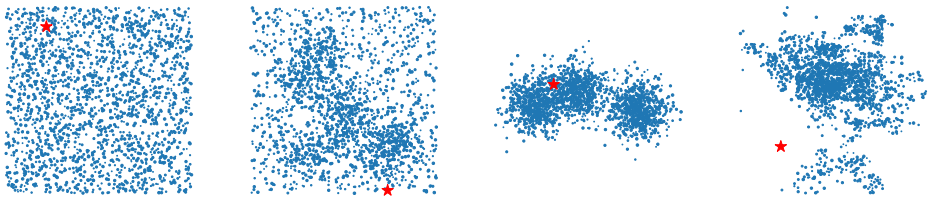

Figure 3: **Instances from $N = 2000$ CVRP distributions**. From left to right: instance from uniform, mixed ($n_c = 7$ cluster centers), clustered ($n_c = 3$ cluster centers), and real-world. The red star is the location of the depot, while blue dots are cities sized proportional to demand.

6.1 Setup

Due to space limitation, we briefly describe experimental setup in the main text and defer full details to Appendix A.2. Given a particular distribution of VRP, we generate separate sets of training, validation, and test problem instances. Unless otherwise stated, our validation and test sets contain 40 instances each. For each problem instance, we generate a rough initial solution by partitioning it into disjoint subsets of cities and briefly running the subsolver on each subset. Due to its generality, we use LKH-3 [10] as the subsolver for all VRP distributions.

We generate the training set by running our iterative framework and selecting subproblems by enumerating the subsolver on all $S \in \mathcal{S}_{k,\text{local}}$. As many subproblems remain unchanged from X to X', we use previously cached subsolutions when possible instead of rerunning the subsolver. While generation times differ depending on several factors, typically it takes less than ten hours with 200 Intel Xeon Platinum 8260 CPUs to generate a training set of 2000 instances.

To avoid training multiple models for different problem sizes N, we train a single model for each VRP distribution with combined data from $N \in \{500, 1000, 2000\}$. Training takes at most 8 hours on a single NVIDIA V100 GPU. To exploit symmetry in problem instances, we apply rotation and flip augmentation at training time. To evaluate trained selectors on a problem instance, we run the iterative selection framework on a single CPU, with disabled multithreading and no GPU.

We select the best hyperparameters via manual search based on validation set performance. We record final results on the test set as the mean and standard error over 5 runs with different random seeds.

Baselines. Consistent with our choice of subsolver, a key baseline from the literature is **LKH-3** [10]. We run the LKH-3 solver on the full problem instance for 30000 local update steps, which takes 2-3 hours on average for $N = 2000$. For fairness, we warm-start LKH-3 with our initialization. We also compare against OR Tools [Perron and Furnon], another open source solver. We set 15 min as the time-out for OR Tools, which converges in all settings.

To compare against previous learning methods, we include AM [14] and NeuRewriter [5]. Most of the previous work do not outperform LKH-3 even on small problem sizes, and learning methods have had more difficulty generalizing to larger instances. In fact, all of the previous methods are trained on problems of size $N \leq 100$, and we have found that they generally do not yield good outcomes on $N \geq 500$, without modifying the architecture and doing extensive retraining. The AM and NeuRewriter results demonstrate the difficulty of scaling up previous learning methods. We do not warm-start OR-Tools, AM, and NeuRewriter because we find empirically that these methods have limited solution capability, and they do not improve from our decent warm-start.

To validate the our subproblem selector's ability to learn a meaningful subproblem-selection order, we design three additional baselines that employ our iterative framework using *hand-crafted heuristics* to select subproblems. The three heuristics that we use are: (1) Random, selecting subproblem S from $\mathcal{S}_{k,\text{local}}$ uniformly; (2) Count-based, which avoids repetitive selections by selecting the subproblem centered at the route whose city nodes have been selected cumulatively the least often in previous steps; and (3) Max Min Distance, which encourages coverage of the entire problem instance by selecting subproblems with the maximum distance from the centroids of previously selected subproblems. We run the heuristic baselines with the same setup as our subproblem selector.

Metrics. We refer to two metrics to compare our method against baseline methods.

Table 1: **Performance and computation time for Uniform CVRP**. For problem instance sizes $N \in \{500, 1000, 2000\}$, we report the objective values (*lower* is better) of our method and baseline methods, averaged across all instances in the test set. Note that the cost is the total distance of routes in the solution. LKH-3 (30k) runs LKH-3 for 30k steps to near convergence, while LKH-3 (95%) runs LKH-3 until 95% of the total improvement has been attained. Random, Count-based, Max Min Distance, and Ours (Short) run until matching LKH-3 (95%) in solution quality with the speedup reported in parentheses, while Ours (long) runs for twice amount time as Ours (Short).

	N = 500		**N = 1000**		**N = 2000**	
Method	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
LKH-3 (95%)	62.00	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	73.60	58s	136.29	2.3min	257.61	8.1min
Random	61.99	71s (3.8x)	120.02	3.2min (5.5x)	234.88	6.4min (8.0x)
Count-based	61.99	59s (4.5x)	120.02	2.1min (8.2x)	234.88	5.3min (10x)
Max Min	61.99	59s (4.5x)	120.02	2.5min (7.0x)	234.89	5.2min (10x)
Ours (Short)	61.99	38s (7.0x)	119.87	1.5min (12x)	234.89	3.4min (15x)
Ours (Long)	**61.70**	76s	**119.55**	3.0min	**233.86**	6.8min

1. **Improvement over method X**: at a specified computation time, the improvement of method Y over method X is the total improvement of method Y minus that of method X.
2. **Speedup over method X**: at a specified solution quality that method X attains, the speedup of method Y over method X is the computation time required for method X to attain the solution quality divided by that for method Y to attain the solution quality.

We often report the numerical speedup over LKH-3 at 95% solution quality (i.e. 95% of the maximum improvement attained by LKH-3) as LKH-3 takes an exorbitant amount of time to attain the last 5% of improvement; reporting the speedup at 100% LKH-3 solution quality would inflate the speedup.

6.2 Uniform CVRP Distribution

As seen in Table 1, our method achieves the best performance for all problem sizes, matching LKH-3's solution quality with more than 7x to 15x less computation time and offering even more improvements with longer computation time. Although we need to evaluate all $R = O(N)$ subproblems of the initial solution in the first step with our subproblem selector, the subsequent per-step computation time of our method after the first step is mostly independent of the problem size N since we only need to evaluate subproblems that have changed.

Running LKH-3 for 30k local update steps achieves superior performance to all previous other heuristic and learning-based baselines. Its solution quality scales well to large problem sizes, yet the solution time is significantly longer. Previous learning-based methods, although fast, result in much worse solution qualities. Our heuristic baselines Random, Count-based, and Max Min Distance demonstrate that our iterative framework, even without the learned subproblem selector, may achieve over 5x to 10x speedup over LKH-3. Nevertheless, our results demonstrate that learning the subproblem selector may offer an additional 1.5x speedup over non-learning heuristics.

In Figure 4, we demonstrate the solution quality of our method and baselines compared to LKH-3. We see that, compared to baseline methods, the learned subproblem selector begets the best solution quality when run for a reasonable amount of time. The improvement of most methods based on our iterative framework converge when run for an exorbitant amount of time; this is unsurprising, as Random and other baselines are eventually able to select all subproblems offering improvements.

In Figure 5, we demonstrate the speedup of our method over LKH-3, comparing to baseline methods. The speedup is often significant even at low levels of solution quality, and improves with higher solution quality. The speedup is not as meaningful past 95% of LKH-3's maximum improvement, as LKH-3 takes an exorbitant amount of time to attain the last 5% of improvement.

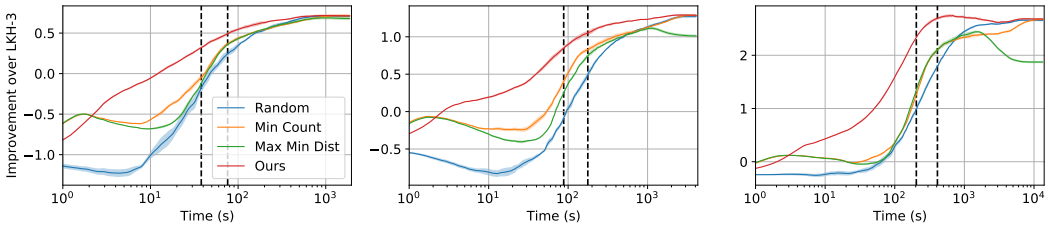

Figure 4: **Improvement over LKH-3 for Uniform CVRP**. The x-axis is the computation time of our method and LKH-3 and extends until LKH-3 has completed 30k steps. The dotted vertical lines represent the computation times of Ours (Short) and Ours (Long) from Table 1.

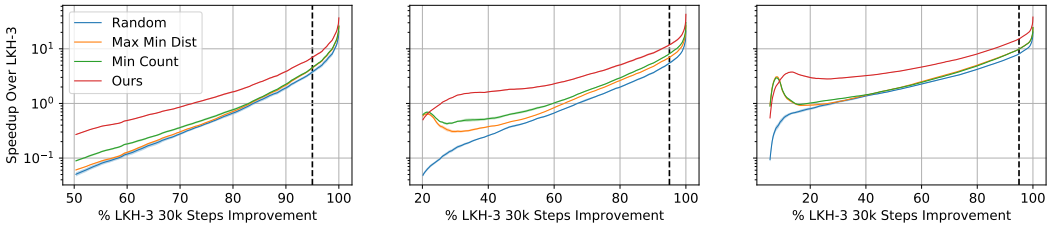

Figure 5: **Speedup over LKH-3 for Uniform CVRP**. The x-axis is the solution quality attained, measured as a percentage of the LKH-3's maximum improvement. The dotted vertical line represents the 95% solution quality which we use to compute the numerical speedup, as mentioned in Table 1.

6.3 Clustered and Mixed CVRP Distributions

We further examine the framework's performance on CVRP distributions with clusters, such as studied in [6, 34]. We generate a clustered instance by first sampling (x, y) locations of n_c cluster centroids then sampling the cities around the centroids. The mixed distribution samples 50% of the cities from a uniform distribution and the rest from around cluster centers. We generate a dataset of clustered and mixed instances for $(N, n_c) \in \{500, 1000, 2000\} \times \{3, 5, 7\}$ and train a single model on the entire dataset. Due to space limitation, we provide more details about the data distribution and full results in the Appendix A.3.

Table 2: **Speedup for** $N = 2000$ **Clustered and Mixed CVRPs** at 95% maximum LKH-3 solution quality.

Setting		$n_c = 3$	$n_c = 5$	$n_c = 7$
Clustered	Ours	26x	18x	25x
	Random	11x	7.5x	9.0x
Mixed	Ours	13x	14x	14x
	Random	6.6x	6.4x	7.6x

Table 2 reports the speedup of our learned subproblem selector and the Random baseline over LKH-3 at the 95% maximum LKH-3 solution quality. We can see that our method offers a 2x speedup over the Random baseline in all settings. Interestingly, we see a larger speedup over LKH-3 for the clustered distribution than that for the mixed distribution and uniform distribution in Table 1.

6.4 Generalization

We study how our subproblem selector generalize to a uniform CVRP distribution with larger problem size $N = 3000$ and to a real-world CVRP distributions, both of which are unseen at training time. The real world CVRP distribution derives from a CVRP dataset [4] consisting of 10 very-large-scale instances on 5 Belgium regions with N ranging from 3000 to 30000 and randomly generated demand distribution. To generate an instance, we subsample the cities in a region without replacement to a smaller size $N = 2000$, while regenerating the demand to have the same distribution as our uniform and cluster training dataset. The original and subsampled dataset are visualized in Appendix A.4.

We apply subproblem selectors trained on uniform and clustered data without finetuning on the new data distributions and report the speedup comparison with the Random baseline in Figure 6. We see that when transferring to $N = 3000$, the subproblem selectors trained on uniform and clustered data

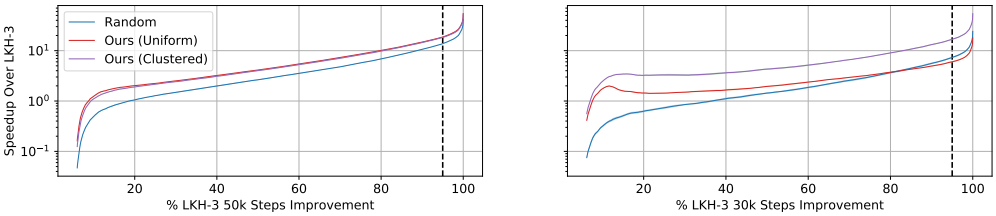

Figure 6: **Speedup in Out-of-distribution CVRPs**. We report the speedup when evaluating our model without finetuning on the $N = 3000$ uniform distribution (left) and $N = 2000$ real-world distribution (right). Ours (Uniform) and Ours (Clustered) are trained on the uniform and clustered CVRP distributions, respectively. Note that LKH-3 is run for 50k steps for the $N = 3000$ instance.

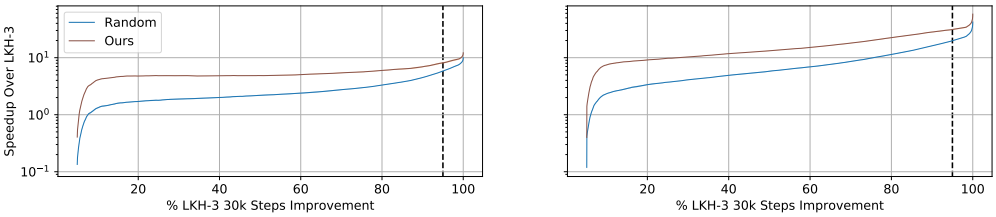

Figure 7: **Speedup in other $N = 2000$ VRP Variants**. We report the speedup when evaluating our method on CVRPTW (left) and VRPMPD (right).

offer similar performance. However, the model trained on the clustered distribution generalizes well to the real-world distribution while the model trained on the uniform distribution fails to generalize, with worse speedup than the Random baseline. These results suggest that the domain variability of the clustered distribution strongly improves generalization performance.

6.5 Other VRP Variants

While our previous experiments focus on varying the distribution of city locations in CVRP, here we consider two VRP variants with uniform city distribution but with additional constraints: CVRPTW [28] and VRPMPD [25]. The former specifies a hard time window constraint for visiting each city, while the latter specifies pick up and delivery constraints in addition to capacity constraint. A detailed description of the variants can be found in Appendix A.5.

Similar to the CVRP distributions, we observe a significant speedup from applying our iterative framework, with additional speedup when using a learned subproblem selector. For CVRPTW, our method offers a 8.2x speedup while our Random baseline offers a 5.9x speedup; for VRPMPD, our method offers a 31x speedup while our Random baseline offers a 20x speedup. We suspect that the time window constraint in CVRPTW is more difficult to learn for the subproblem selector, as it imposes strict orderings on the order of city visitations.

7 Conclusion

This paper presents a learning-based framework for large VRPs, which learns which subproblems to delegate to a subsolver. Spatial locality in VRPs allows us to learn the subproblem selector over a reduced subproblem space. The proposed method outperforms the state-of-art VRP solver LKH-3 on problems of sizes up to 3000, with an order of magnitude less computation time. As future work, we plan to apply our method to other combinatorial optimization problems with spatial locality. We believe that our learning framework can serve as a powerful technique for both the learning and operations research communities to scale up combinatorial optimization solvers.

Negative Social Impact. Enabling more efficient solutions of large-scale VRPs may exhibit negative externalities such as inducing additional traffic from delivery vehicles and centralizing services that pose a stronger competition to brick-and-mortar retail.

References

[1] Aarts, E., Aarts, E. H., and Lenstra, J. K. (2003). *Local search in combinatorial optimization*. Princeton University Press.

[2] Ahn, S., Seo, Y., and Shin, J. (2020). Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pages 134–144. PMLR.

[3] Alvim, A. C. and Taillard, E. D. (2013). Popmusic for the world location-routing problem. *EURO Journal on Transportation and Logistics*, 2(3):231–254.

[4] Arnold, F., Gendreau, M., and Sörensen, K. (2019). Efficiently solving very large-scale routing problems. *Computers & Operations Research*, 107:32–42.

[5] Chen, X. and Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[6] Christofides, N., Mingozzi, A., and Toth, P. (1979). Loading problems. *N. Christofides and al., editors, Combinatorial Optimization*, pages 339–369.

[7] Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.

[8] Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

[9] Gurobi Optimization, L. (2021). Gurobi optimizer reference manual.

[10] Helsgaun, K. (2017). An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. `http://akira.ruc.dk/~keld/research/LKH-3/`.

[11] Hottung, A. and Tierney, K. (2020). Neural large neighborhood search for the capacitated vehicle routing problem. In *24th European Conference on Artificial Intelligence (ECAI 2020)*.

[12] Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer.

[13] Khalil, E. B., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*.

[14] Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

[15] Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., and Min, S. (2020). Pomo: Policy optimization with multiple optima for reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198. Curran Associates, Inc.

[16] Lalla-Ruiz, E. and Voss, S. (2016). Popmusic as a matheuristic for the berth allocation problem. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):173–189.

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

[18] Laurent, M., Taillard, É. D., Ertz, O., Grin, F., Rappo, D., and Roh, S. (2009). From point feature label placement to map labelling. In *Proceedings, metaheuristic international conference (MIC'09), Hamburg*. Citeseer.

[19] Lenstra, J. K. and Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

[20] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.

[21] Lu, H., Zhang, X., and Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*.

[22] Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

[23] Nazari, M., Oroojlooy, A., Snyder, L., and Takac, M. (2018). Reinforcement learning for solving the vehicle routing problem. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Perron and Furnon] Perron, L. and Furnon, V. Or-tools.

[25] Salhi, S. and Nagy, G. (1999). A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *Journal of the operational Research Society*, 50(10):1034–1042.

[26] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

[27] Sivanandam, S. and Deepa, S. (2008). Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer.

[28] Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265.

[29] Song, J., Lanka, R., Yue, Y., and Dilkina, B. (2020). A general large neighborhood search framework for solving integer linear programs. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20012–20023. Curran Associates, Inc.

[30] Taillard, É. D. (2003). Heuristic methods for large centroid clustering problems. *Journal of heuristics*, 9(1):51–73.

[31] Taillard, É. D., Gambardella, L. M., Gendreau, M., and Potvin, J.-Y. (2001). Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16.

[32] Taillard, É. D. and Voss, S. (2002). Popmusic—partial optimization metaheuristic under special intensification conditions. In *Essays and surveys in metaheuristics*, pages 613–629. Springer.

[33] Taillard, É. D. and Voß, S. (2018). *POPMUSIC*, pages 687–701. Springer International Publishing, Cham.

[34] Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., and Subramanian, A. (2017). New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858.

[35] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[36] Vinyals, O., Fortunato, M., and Jaitly, N. (2015). Pointer networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

[37] Voudouris, C. and Tsang, E. P. (2003). Guided local search. In *Handbook of metaheuristics*, pages 185–218. Springer.

[38] Wu, C., Shankari, K., Kamar, E., Katz, R., Culler, D., Papadimitriou, C., Horvitz, E., and Bayen, A. (2016). Optimizing the diamond lane: A more tractable carpool problem and algorithms. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1389–1396. IEEE.

[39] Yolcu, E. and Poczos, B. (2019). Learning local search heuristics for boolean satisfiability. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.2. We discuss our method gets slightly less speedup on CVRPTW than CVRP or VRPMPD; we also discuss our learned subproblem-selection order at convergence are the same as Random and Min Count order, which can be seen as a potential limitation, but we provide justification in the same Section (6.2) as well.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the negative social impact paragraph in Section 7 (Conclusion)
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the link to our code and instructions as an URL at the end of the Appendix. We plan to release the code, data, and instructions on github upon paper acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We briefly mention the training details in Section 6.1 of the main paper, and we put the full details in Appendix A.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We plot error bars on all of our experiment figures in Section 6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 6.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use the LKH-3 VRP solver as a part of our component. We cite the creator as [10] in our paper. We compare with existing learning and heuristic baseline using their code. We cite the creators in our paper, and provide exerpeiment setup (including github links to the repos) in the supplementary material.
   (b) Did you mention the license of the assets? [N/A] All the assets (code and data) we use are open source.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We generate synthetic data for most of our experiments. The only real dataset in Section 6.4 is open source.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]