Bangladesh University of Engineering and Technology

Course: CSE-462 (Algorithm Engineering Sessional)
Project: Neural Combinatorial Optimization

Submitted by:
Md. Ashrafur Rahman Khan (1905005)
Md. Roqunuzzaman Sojib (1905067)
Mashroor Hasan Bhuiyan (1905069)
Mahir Labib Dihan (1905072)
Wasif Jalal (1905084)

Date of Submission: 28.12.2024

Contents

Chapter 1

Introduction

Combinatorial optimization problems are fundamental to numerous real-world applications, such as logistics, scheduling, and network design. These problems require finding an optimal solution from a finite set of possibilities, constrained by feasibility requirements. Traditional solution methods, including exact algorithms and heuristics, often struggle with scalability as problem sizes grow. This limitation has driven the development of innovative approaches like Neural Combinatorial Optimization (NCO), which leverages advances in machine learning to tackle these challenges.

NCO combines neural networks with algorithmic principles to learn solution strategies directly from data, bypassing the reliance on handcrafted heuristics. Using supervised or reinforcement learning, NCO models are trained to approximate optimal solutions or maximize problem-specific rewards. Reinforcement learning-based techniques, such as Policy Optimization with Multiple Optima (POMO), have demonstrated strong performance on routing problems, offering scalable and effective solutions.

The Capacitated Vehicle Routing Problem (CVRP) is a classic combinatorial optimization problem and an ideal candidate for exploring NCO. It involves designing routes for vehicles to service customers while minimizing costs, adhering to vehicle capacity limits, and ensuring each customer is served exactly once. CVRP's complexity and practical significance make it a valuable testbed for NCO methods, enabling insights into how these models address real-world optimization challenges.

This report focuses on applying NCO to the CVRP, exploring how neural networks can learn effective heuristics for solving this problem. The report is structured as follows:

- **Chapter 2** provides an overview of traditional exact algorithms used for solving combinatorial problems, highlighting their strengths and limitations.
- **Chapter 3** discusses approximation algorithms, which trade off solution quality for improved computational efficiency.
- **Chapter 4** delves into heuristic methods, both constructive and improvement-based, showcasing how these methods build and refine solutions iteratively.
- **Chapter 5** introduces metaheuristics, a class of advanced heuristics inspired by natural processes and optimization theory, which are often used to tackle large-scale CVRP instances.
- **Chapter 6** focuses on Neural Combinatorial Optimization, detailing the methodologies, architectures, and learning paradigms used to apply NCO to CVRP. This chapter also covers experimental evaluations of NCO techniques, including supervised learning methods, reinforcement learning frameworks like POMO, and techniques for improving model performance using instance augmentation and beam search.

By integrating traditional optimization techniques with cutting-edge neural models, this report provides a comprehensive exploration of the CVRP and its modern-day solutions. Our study demonstrates the potential of NCO as a transformative approach to solving combinatorial optimization problems, bridging the gap between theoretical research and practical applications.

Chapter 2

Exact Algorithms

We briefly review the most recent works proposing exact algorithms for the CVRP; all of them are based on the combination of column and cut generation.

Fukasawa et al. [**15**] presented a BCP algorithm having the following features:

- The columns are associated with the q-routes without k-cycles, a relaxation of the elementary routes that allow multiple visits to a customer, on the condition that at least k other costumers are visited between successive visits. The separated cuts are the same used in previous Branch-and-Cut algorithms over the two-index CVRP formulation. Those cuts are robust with respect to q-route pricing.

- If the column generation at the root node is found to be too slow, the algorithm automatically switches to a Branch-and-Cut.

All benchmark instances from the literature with up to 135 vertices could be solved to optimality, a significant improvement over previous methods.

Baldacci, Christofides, and Mingozzi [**3**] presented an algorithm based on column and cut generation with the following features:

- The columns are associated with elementary routes. Besides cuts for the two-index formulation, Strengthened Capacity Cuts and Clique Cuts are separated. Those latter cuts are effective but non-robust; they make the pricing harder.

- A sequence of cheaper lower bounding procedures produces good estimates of the optimal dual variable values. The potentially expensive pricing is only called in the last stage, with the dual variables bounded to be above the estimates, obtaining convergence (to a bound slightly below the theoretical optimal) in fewer iterations.

- Instead of branching, the algorithm finishes in the root node (therefore, it is not a BCP) by enumerating all elementary routes with reduced cost smaller than the duality gap. A set-partitioning problem containing all those routes is then given to a Mixed-Integer Programming (MIP) solver.

The algorithm could solve almost all instances solved in [15], usually taking much less time. However, the exponential nature of some algorithmic elements, in particular the route enumeration, made it fail on some instances with many customers per vehicle.

Pessoa, Poggi de Aragão, and Uchoa [27] presented some improvements over Fukasawa et al. [15]:

- Cuts from an extended formulation with capacity indices were also separated. Those cuts do not change the complexity of the pricing of q-routes by dynamic programming.

- The idea of performing elementary route enumeration and MIP solving to finish a node was borrowed from [3]. However, in order to avoid a premature failure when the root gap is too large, it was hybridized with traditional branching.

Baldacci, Mingozzi, and Roberti [4] improves upon Baldacci, Christofides, and Mingozzi [3] by the introduction of the following elements:

- The *ng*-routes, a newly proposed relaxation that is more effective than the q-routes without k-cycles, are used in the earlier bounding procedures and also to accelerate the pricing/enumeration of elementary routes. This latter part of the algorithm is also enhanced by considering multiple dual solutions. A route having reduced cost larger, with respect to any dual solution, than the current duality gap can be discarded.

- Subset Row Cuts and Weak Subset Row Cuts, which have less impact on the pricing with respect to Clique Cuts, are separated.

The resulting algorithm is not only faster on average, but it is much more stable than the algorithm in [3], being able to solve even some instances with many customers per vehicle.

Contardo [10] introduced new twists on the use of non-robust cuts and on route enumeration:

- The columns are associated with q-routes without 2-cycles, a relatively poor relaxation. The partial elementarity of the routes is enforced by non-robust Strong Degree Cuts. Robust cuts from two-index formulation and non-robust Strengthened Capacity Cuts and Subset Row Cuts are also separated.

- The enumeration of elementary routes is directed to a pool of columns. As soon as the duality gap is sufficiently small to produce a pool with reasonable size (a few million routes), the pricing starts to be performed by inspection. From this point, an aggressive separation of non-robust cuts can be performed, leading to very small gaps.

The reported computational results are very consistent. In particular, a hard in- stance from the literature, M-n151-k12 (151 vertices, 12 vehicles), was solved to optimality, setting a new record.

Røpke [32] went back to robust BCP. The main differences from [15] are the following:

- Instead of q-routes without k-cycles, the more effective ng-routes are used.

- A sophisticated and aggressive strong branching is performed, drastically reducing the average size of the enumeration trees.

The overall results are comparable with the results in [10] and [4]. A long run of that algorithm could also solve M-n151-k12.

Contardo and Martinelli [11] improved upon Contardo [10]:

- Instead of q-routes without 2-cycles, ng-routes are used. Moreover, the performance of the dynamic programming pricing was enhanced by the use of the DSSR technique [31].

- Edge variables are fixed by reduced costs, using the procedure proposed in [19].

Finally, **Pecin et al. [26]** proposed a BCP that incorporates elements from all the previous algorithms, usually enhanced and combined with new elements:

- The most important original contribution is the introduction of the limited memory Subset Row Cuts. They are a weakening of the traditional Subset Row Cuts that can be dynamically adjusted, making them much less costly in the pricing, and yet without compromising their effectiveness.
- It uses capacity indices [27] in its underlying formulation. This allows a fixing of variables by reduced costs that is superior to fixing in [19].
- The columns in the BCP are associated with ng-routes. The dynamic programming pricing uses bidirectional search that differs a little from that proposed in [30] because the concatenation phase is not necessarily performed at half of the capacity. A column generation stabilization by dual smoothing [28] may also be employed.
- The BCP hybridizes branching with route enumeration. Actually, it performs an aggressive hierarchical strong branching.
- When the addition of a round of non-robust cuts makes the pricing too slow, the BCP performs a rollback. The offending cuts are removed even if the lower bound of the node decreases.

The algorithm could solve all the classical instances from the literature with up to 200 vertices in reasonable times, including the hard instances M-n200-k17 and M-n200-k16.

Chapter 3

Approximation Algorithms

For general metrics, Haimovich and Kan [16] considered a simple heuristic, called tour partitioning, which starts from a TSP tour and then splits it into tours of size at most Q by making back-and-forth trips to r at certain points along the TSP tour. They showed this gives a $(1 + (1 - 1/Q)\alpha)$-approximation for splittable CVRP, where α is the approximation ratio for TSP. Essentially the same algorithm yields a $(2+(1-2/Q)\alpha)$-approximation for unsplittable CVRP [2]. These stood as the best-known bounds until recently, when Blauth et al. [8] showed that given a TSP approximation α, there is an $\epsilon > 0$ such that there is an $(\alpha + 2 \cdot (1 - \epsilon))$-approximation algorithm for CVRP. For $\alpha = 3/2$, they showed $\epsilon > 1/3000$. They also describe a $(\alpha + 1 - \epsilon)$-approximation algorithm for unit demand CVRP and splittable CVRP. Friggstad et al. [14] presented two approximation algorithms for unsplittable CVRP: a combinatorial $(\alpha + 1.75)$-approximation, where α is the approximation factor for the Traveling Salesman Problem, and an approximation algorithm based on LP rounding with approximation guarantee $\alpha + ln(2) + \delta \approx 3.194 + \delta$ in $n^{O(1/\delta)}$ time. Both approximations can further be improved by a small amount when combined with recent work by Blauth, Traub, and Vygen (2021).

Approximation by Friggstad et al. is the current best for the general case. However, there have been improvements for special cases in recent years. CVRP remains APX-hard in general metrics in this case but is polynomial-time solvable on trees. There exists a PTAS for CVRP in the Euclidean plane as shown by Khachay et al. [21]. A PTAS for planar graphs was given by Becker et al. [7] and a QPTAS for planar and bounded-genus graphs was then given by Becker et al. [5]. A PTAS for graphs of bounded highway

dimension and an exact algorithm for graphs with treewidth tw with running time $O(n^{twQ})$ was shown by Becker et al. [6]. Cohen-Addad et al. [9] showed an efficient PTAS for graphs of bounded-treewidth, an efficient PTAS for bounded highway dimension, an efficient PTAS for bounded genus metrics and a QPTAS for minor-free metrics.

Chapter 4

Heuristics

Exact methods have a guarantee of solution optimality and are more theoretically sound. However, it is difficult for them to handle the increasingly complex modern real-world vehicle routing problems efficiently. In contrast, heuristics provide a high-quality solution at a reasonable computational cost and have good generalization ability, and thus have gained popularity among both researchers and practitioners. The heuristics are categorized into two main categories: 1) Constructive heuristics, 2) Improvement Heuristics

4.1 Constructive Heuristics

Constructive heuristics construct routing solutions from zero following some fixed empirical procedures. They build a solution quickly, yet there is usually a gap between its result and the optimal one. The existing con- structive heuristics can be summarized into four algorithm frameworks: 1) nearest neighbor method, 2) insert method, 3) saving method, and 4) sweep method.

4.1.1 Nearest Neighbor Method

The routes can be built either sequentially or parallelly. In sequential route building, a route is extended by greedily adding the nearest feasible unrouted customer with the depot as the starting node. A new route is initialized from the depot when no customer can be added. The shortcoming of sequential route building is that the last vehicle usually has a lower loading ratio compared to other routes. The shortcoming can be alleviated if the routes are

constructed in parallel. In parallel route building, the number of vehicles K is set beforehand and the K routes are extended in parallel. In each iteration, each route is extended with the closest unrouted customer, which means K customers will be added. The process is iteratively performed until all the customers are visited. If customers can not be added to the K routes, then a new route will be created following the sequential route-building strategy. The time complexity of the nearest neighbor method with n customers is $O(n^2)$, The algorithm iterates n times and takes $O(n)$ to find the nearest neighbor in each iteration.

4.1.2 Insert Method

It initializes some empty routes and inserts the unrouted customers one by one into the routes but not necessarily into the end of each route, which is the solution in the nearest neighbor method. In each iteration, the insertion that has the minimum cost is performed. The worst time complexity of the insert method is $O(n^3)$. Similar to the nearest neighbor method, the insert method iterates n times. However, there are two layers of minimization problem in each iteration: first, to find the minimal insertion cost for each unrouted customer and then find the unrouted customer with the cheapest minimal insertion cost.

4.1.3 Saving Method

It starts with an initial solution, in which a different route serves each customer, and iteratively combines the short routes into longer routes with lower overall costs. The procedure can be performed in parallel or in sequence. In its parallel version, the method iteratively combines two routes with the endpoints i and j, which produces the maximum feasible distance saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. In the sequential version, each route is considered in turn. One route is iteratively extended by feasible saving operation until no such feasible merge can be used. The time complexity of a straightforward implementation of the saving method is $O(n^3)$. The algorithm starts with n routes. It takes n merge operations (iterations) and each merge operation is decided at the cost of $O(n^2)$ if every maximum feasible distance saving $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ is checked or rechecked in each iteration. To reduce the complexity, the $n(n-1)/2$ combinations corresponding to $n(n-1)/2$ possible

mergers can be computed and sorted at the beginning of the algorithm. The sort can be finished in $O(n^2 log_2 n)$ using a fast sorting algorithm.

4.1.4 Sweep Method

The algorithm first sets the depot as the origin and sorts the nodes according to the polar angle. In its basic version, the node is added to the route circularly. A new route will be created if the insertion is infeasible. Another approach is a cluster-first, route-second method. All the customers are clustered into several clusters according to the polar angle and a TSP is solved in each cluster. The performance of the sweep method can be greatly affected by the location of the depot. A poor result will be generated when the depot is off-centered. In addition, the last route often has a lower loading ratio because the clusters are built one by one. The time complexity of the basic sweep method is $O(n)$ because all the required computing is adding the next unrouted customer to the previous route according to the polar angle. In more advanced versions, the time complexity mainly depends on the method used for the TSPs.

4.2 Improvement Heuristics

Improvement heuristics explore the neighborhood of the incumbent so- lution to achieve an objective improvement. They can quickly converge to the local optimal and thus are efficient at solving large-scale routing problems. There are two widely recognized classes of improvement heuristics: intra-route and inter-route. The difference between the two classes is the neighborhood structure. The former searches inside a single route, while the latter involves multiple routes.

4.2.1 Intra-route Method

Intra-route improvement heuristics explore the neighborhood involving only one route. Most of them originated from the local search operators for TSP. For instance, among the simplest ones, one customer can be relocated to a different position in the same route, or two customers in one route can be exchanged. The descriptions for some heuristics are given as follows:

- *Relocate*: Relocate selects one route from the solution and relocates one node to another position in the route. The time complexity of finding the best relocation solution is $O(n^2)$.

- *Exchange*: Exchange selects one route and exchanges the positions of two nodes in the route. The time complexity of finding the best relocation solution is $O(n^2)$.

- *2-opt,3-opt,λ-opt*: 2-opt first selects one route and then reconnects the ends of two edges in the route. In other words, it reverses a sequence of nodes within the route. The time complexity of finding the best reconnection is $O(n^2)$. The 2-opt can be regarded as a subset of 3-opt, which reconnects three edges with a time complexity of $O(n^3)$. λ-opt further generalizes them to considering λ edges.

- *Or-exchange*: Or-exchange selects a sequence of nodes with a preset length and relocates it to another position in the same route. The or-exchange can be regarded as a subset of 3-opt. Different from 3-opt, it only needs to specify the location of the start (or end) node of the sequence and the corresponding destination in the route. Therefore, the time complexity of finding the best exchange is $O(n^2)$.

- *GENI*: GENI selects one route and a subset of three vertices. For a vertex v not yet on the route, it implements the least cost insertion considering the two possible orientations of the tour and the two insert types. The time complexity of each insertion is $O(np^4 + n^2)$ when the neighborhood size is p.

4.2.2 Inter-route Method

Inter-route heuristics involve local searches across multiple routes. Many of them are extensions of intra-route counterparts. For example, insert and swap are the extension of relocate and exchange, respectively. The former removes a customer from one route and reinserts it into another route. The latter swaps two customers from different routes. Some representative inter-route improvement heuristics are introduced as follows:

- *2-opt**: 2-opt* selects two routes and reconnects the end of two edges [218]. The reconnection with the best cost reduction is chosen. If the

cost of the selection of two routes can be neglected, its time complexity is the same as 2-opt.

- *Insert*: Insert selects one node and inserts it into another position. Similar to the Relocation operator, the time complexity is $O(n^2)$.
- *Swap*: Swap selects two nodes and swaps their locations [210]. Like the Exchange operator, the time complexity is $O(n^2)$.
- *CROSS*: CROSS exchanges two customer sequences (one of the two sequences can be empty). It generalizes the three mentioned operators: 2-opt*, Insert, and Swap. The time complexity is at most $O(n^4)$ and $O(\lambda^2 n^2)$ when the size of the sequence is limited by a value λ.
- λ-*interchange*: λ-interchange further generalizes CROSS. It allows exchanging any set of less than λ nodes between two routes. The set can be non-consecutive, empty, and reversed during the reinsert.

Chapter 5

Metaheuristics

Metaheuristics are presented in a more general and high-level way, in contrast to constructive heuristics and improvement heuristics, which are problem-dependent and attempt to exploit the feature and structure of the target problem. According to the population management strategy, we classify metaheuristics into two categories: 1) single-solution-based and 2) population-based method. The widely-used metaheuristics in vehicle routing includes simulated annealing (SA), tabu search (TS), iterated local search (ILS), large neighborhood search (LNS), genetic algorithm (GA), ant colony optimization (ACO), memetic algorithm (MA). These algorithms are among the top-used algorithms in vehicle routing as reviewed in [12]. Among them, GA received the most studies, followed by TS, LNS, and ACO. The number of publications on LNS and MA is growing rapidly in the last decade. Undoubtedly, these algorithms are very efficient in solving any complex optimization problem but they also come with the constraints of slow or premature convergence. Generally, to overcome these constraints, hybrid methods are used. Numerous metaheuristic approaches have been used to solve CVRP, which despite their modest computational capacity, have been proven to be quite effective.

Prins [29] was the first to propose a hybrid genetic algorithm capable of solving vehicle routing problem which was tested on 34 benchmark instances. Notably, prior to the emergence of this hybrid genetic algorithm, genetic algorithms had earlier been utilized to address the CVRP. However, its performance consistently lagged behind compared to other metaheuristic methods. The hybridization of GA with a local search technique by Prins improved the solution accuracy and prevented it from premature convergence. The hybrid GA suffers from slow convergence speed.

Nazif & Lee [25] proposed an optimized crossover operator for solving CVRP utilizing genetic algorithm. It was examined on 27 benchmark instances.

Teoh and his collaborators [35] developed a local search-based improved differential evolution technique (DELS) which was experimented on 88 well-known benchmark instances. Three different local search moves were employed: swapping, dropping one point, and flipping. This implementation improved the exploration and exploitation capability of DELS.

Hosseinabadi et al. [18] formulated a new CVRP-solving method based on gravity emulation local search (CVRP_GELS). The CVRP_GELS algorithm was designed utilizing two of the four basic parameters of gravitational force and velocity in physics based on the concepts of random search and searching agents, which is a collection of masses interacting with each other according to newtonian gravity and the laws of motion. This method was examined on 175 benchmark instances. One of the merits of this algorithm is its low execution time. The algorithm's performance was less satisfactory for small-scale instances but proved highly effective for larger-scale ones.

Faiz et al. [13] devised a mechanism known as PVNS-ASM to handle capacitated vehicle routing problem by integrating perturbation-based variable neighborhood search with an adaptive selection approach to increase population diversity, enhance exploration and exploitation capabilities, and avoid premature convergence to locally optimal solutions. It was examined on 21 benchmark instances.

Sbai et al. [34] constructed a hybrid metaheuristic HGA-VNS. This hybrid approach combines the strengths of genetic algorithm in exploration and the potent exploitation capabilities of variable neighborhood search (VNS). The algorithm outperformed majority of the state-of-the-art methods after a series of tests that involved roughly around 186 instances.

Altabeeb et al. [1] put forth a hybrid firefly algorithm termed as CVRP-FA. To increase the quality of the solution and hasten convergence, they combined 2h-opt and improved 2-opt algorithms. Additionally, partially mapped crossover as well as two mutation operators were also employed. The effectiveness of this technique was examined on 82 well-known benchmark instances.

Sajid et al. [33] developed an NSGA-II-based routing algorithm using a new tour best cost crossover operator to solve bi-objective CVRP. The

algorithm aimed at improving solution quality, avoiding being entrapped in local optima, and accelerating convergence speed. It was evaluated on 88 benchmark instances.

Jiang et al. [**20**] designed a relevance matrix-based evolutionary algorithm, termed as RMEA, to solve CVRP. To evolve the population, a relevance matrix-based crossover operator was designed, and a relevance matrix-based diversity preservation strategy was put forth as a method to increase population diversity. The algorithm was examined on 37 benchmark instances.

Khoo and Mohammad [**22**] have also devised a hybrid ruin recreated genetic algorithm HRRGA to solve multi-objective vehicle routing problem with time windows.

Vidal et al. [**36**] developed HGS-CVRP. Hybrid Genetic Search (HGS) combines elements of genetic algorithms (a global search strategy) with local search techniques to refine solutions further

Similarly, other researchers have also devised different techniques for solving capacitated vehicle routing problem.

Chapter 6

Neural Combinatorial Optimization

Neural Combinatorial Optimization (NCO) is an emerging framework that leverages deep learning to solve complex combinatorial optimization problems. Unlike traditional optimization methods, NCO integrates neural networks to learn representations and decision-making strategies directly from data. This paradigm enables generalization across problem instances, scalability to large problem sizes, and adaptability to varied problem structures. Based on the deep learning paradigm, NCO can broadly be categorized into two approaches: supervised learning and reinforcement learning.

6.0.1 Overview

In supervised learning, NCO models are trained on labeled datasets consisting of problem instances and their corresponding optimal solutions. Let $x \in \mathbb{R}^d$ denote the input features of a combinatorial problem instance and $y \in \mathbb{R}^k$ represent the optimal solution. The goal is to learn a function $f_\theta(x)$ parameterized by a neural network θ, such that $f_\theta(x) \approx y$. The training process minimizes a loss function $L(y, f_\theta(x))$, typically the mean squared error or cross-entropy loss. For instance, in the Capacitated Vehicle Routing Problem (CVRP), x could encode customer locations and demands, and y would represent the optimal route. A common architecture involves an encoder-decoder framework, where the encoder generates a fixed-size embedding for x, and the decoder sequentially predicts components of y.

Reinforcement learning-based NCO methods aim to maximize a reward signal $R(y)$ that quantifies the quality of the solution y. Here, the model interacts with an environment representing the problem, taking actions a_t at each step t to construct y. The probability of taking an action is governed by a policy $\pi_\theta(a_t|s_t)$, where s_t is the state at time t. The objective is to optimize θ to maximize the expected reward:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T} R(a_t)\right].$$

Policy gradient methods, such as REINFORCE, are commonly used to update θ via the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[R(y)\nabla_\theta \log \pi_\theta(y)\right].$$

Techniques like Policy Optimization with Multiple Optima (POMO) enhance the reinforcement learning framework by reducing biases from initial states and enabling efficient exploration of multiple optima.

6.1 Supervised Learning

6.1.1 Literature Review

Supervised learning approaches in Neural Combinatorial Optimization (NCO) aim to train models using labeled datasets containing input-output pairs of problem instances and their optimal solutions. These methods have been pivotal in demonstrating the ability of neural networks to generalize and approximate high-quality solutions for combinatorial problems.

Luo et al. (2023) proposed the Heavy Decoder architecture, a supervised learning framework specifically designed for routing problems like the Capacitated Vehicle Routing Problem (CVRP)[24]. Their model employs a lightweight encoder that processes input features and a computationally intensive decoder that generates solutions sequentially. The heavy decoder leverages multi-head attention mechanisms to capture dependencies between customer nodes, significantly enhancing solution quality. This architecture achieved state-of-the-art performance on large-scale CVRP instances, demonstrating strong generalization across problem sizes.

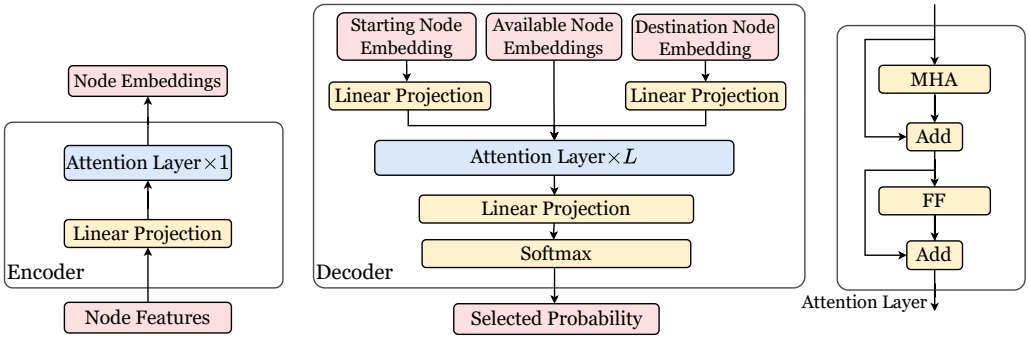

Figure 6.1: The structure of our proposed LEHD model, which has a single-layer light encoder and a heavy decoder with L attention layers.

Supervised learning provides a strong foundation for NCO by utilizing optimal solutions to guide model training. These methods have proven particularly effective for problems with structured datasets, offering a pathway to scalable and high-performance combinatorial optimization solutions.

6.1.2 Light Encoder and Heavy Decoder

We introduce a novel NCO model with the Light Encoder and Heavy Decoder (LEHD) structure to tackle the critical issue regarding large-scale generalization. The LEHD-based model consists of a light encoder and a heavy decoder, as shown in Figure 6.1. The light encoder has one attention layer, and the heavy decoder has L attention layers.

Encoder

For a problem instance $\mathbf{S}$ with n node features $(\mathbf{s}_1, \ldots, \mathbf{s}_n)$ (e.g., the coordinates of n cities), a constructive model parameterized by $\boldsymbol{\theta}$ generates the solution in an autoregressive manner, i.e., it constructs the solution by selecting nodes one by one. The light encoder transforms each node features $\mathbf{s}_i$ to its initial embedding $\mathbf{h}_i^{(0)}$ through a linear projection. The initial embeddings $\{\mathbf{h}_1^{(0)}, \ldots, \mathbf{h}_n^{(0)}\}$ are then fed into one attention layer to get the node embedding matrix $H^{(1)} = (\mathbf{h}_1^{(1)}, \ldots, \mathbf{h}_n^{(1)})$.

Attention Layer

The attention layer comprises two sub-layers: the multi-head attention (MHA) sub-layer and the feed-forward (FF) sub-layer vaswani2017attention. Different from generic NCO models such as AM kool2018attention, the normalization is removed from our model to enhance generalization performance (see Appendix **??**). Let $H^{(l-1)} = (\mathbf{h}_1^{(l-1)}, \ldots, \mathbf{h}_n^{(l-1)})$ be the input of the l-th attention layer for $l = 1, \ldots, L$, the output of the attention layer in terms of the i-th node is calculated as:

$$\begin{aligned} \hat{\mathbf{h}}_i^{(l)} &= \mathbf{h}_i^{(l-1)} + \mathrm{MHA}(\mathbf{h}_i^{(l-1)}, H^{(l-1)}), \\ \mathbf{h}_i^{(l)} &= \hat{\mathbf{h}}_i^{(l)} + \mathrm{FF}(\hat{\mathbf{h}}_i^{(l)}). \end{aligned} \tag{6.1}$$

We denote this embedding process as $H^{(l)} = \mathrm{AttentionLayer}(H^{(l-1)})$.

Decoder

The decoder sequentially constructs the solution in n steps by selecting node by node. In the t-th step for $t \in \{1, \ldots, n\}$, the current partial solution can be written as $(x_1, \ldots, x_{t-1})$, the first selected node's embedding is denoted as $\mathbf{h}_{x_1}^{(1)}$, the embedding of the node selected in the previous step is denoted as $\mathbf{h}_{x_{t-1}}^{(1)}$, and the embeddings of all available nodes is denoted by $H_a = \{\mathbf{h}_i^{(1)} \mid i \in \{1, \ldots, n\} \setminus \{x_1, \ldots, x_{t-1}\}\}$. Since the decoder has L attention layers, the t-th construction step of LEHD can be described as:

$$\begin{aligned} \widetilde{H}^{(0)} &= \mathrm{Concat}(W_1 \mathbf{h}_{x_1}^{(1)}, W_2 \mathbf{h}_{x_{t-1}}^{(1)}, H_a), \\ \widetilde{H}^{(1)} &= \mathrm{AttentionLayer}(\widetilde{H}^{(0)}), \\ &\cdots \\ \widetilde{H}^{(L)} &= \mathrm{AttentionLayer}(\widetilde{H}^{(L-1)}), \\ u_i &= \begin{cases} W_O \widetilde{\mathbf{h}}_i^{(L)}, & i \neq 1 \text{ or } 2 \\ -\infty, & \text{otherwise} \end{cases}, \\ \mathbf{p}^t &= \mathrm{softmax}(\mathbf{u}), \end{aligned} \tag{6.2}$$

where W_1, W_2, W_O are learnable matrices. The matrices W_1 and W_2 are used to re-calculate the embeddings of the starting node (i.e., the node selected in the previous step) and the destination node (i.e., the node selected in the first step), respectively. The matrix W_O is used to transform the relation-denoting embedding matrix $\widetilde{H}^{(L)}$ into a vector $\mathbf{u}$ for the purpose of calculating

the selection probability $\mathbf{p}^t$. The most suitable node x_t is selected based on $\mathbf{p}^t$ at each decoding step t. Finally, a complete solution $\mathbf{x} = (x_1, \ldots, x_n)$ is constructed by calling the decoder n times.

The heavy decoder dynamically re-embeds the embeddings of the starting node, destination node, and available nodes via L attention layers, thus updating the relationships among the nodes at each decoding step. Such a dynamic learning strategy enables the model to adjust and refine its captured relationships between the starting/destination and available nodes. In addition, as the size of the nodes varies during the construction steps, the model tends to learn the scale-independent features. These good properties enable the model to dynamically capture the proper relationships between the nodes, thereby making more informed decisions in the node selection on problem instances of various sizes.

6.1.3 Codebase

The GitHub link for the official code is: `https://github.com/CIAM-Group/NCO_code`. We have modified the code to improve the performance. Github link: `https://github.com/mahirlabibdihan/NCO_code`. An overview of the main modules is given below:

- VRPTrainer: Main training loop.
- VRPTester: Main testing loop.
- VRPModel: Encoder-Decoder Model Architecture.
- VRPEnv: Maintains state of a batch.

6.1.4 Dataset

The standard data generation procedure of the previous work [23] was followed to generate CVRP datasets. The training sets consist of one million instances, each with 100 nodes. Optimal solutions are obtained using HGS-CVRP [36]. The test set includes 10,000 instances with 100 (CVRP100) nodes and 128 instances for each of 200 (CVRP200), 500 (CVRP500), and 1000 (CVRP1000) nodes. Optimal solutions are obtained using LKH3 [17]. The training and test datasets can be downloaded from Google Drive[1] or

[1] `https://drive.google.com/drive/folders/1LptBUGVxQlCZeWVxmCzUOf9WPlsqOROR?usp=sharing`

Baidu Cloud[2].

6.1.5 Methodology

During the inference phase, LEHD model employs a greedy step-by-step approach to construct the whole solution. The tour generated via greedy search could be not optimal and contain multiple suboptimal local segments (e.g., partial solutions). Therefore, rectifying these suboptimal segments during the inference phase can effectively enhance the overall solution quality. For this, Random Re-Construct(RRC) is proposed. RRC randomly samples a partial solution from the initial solution and restructures it to obtain a new partial solution. If the new partial solution is superior, it replaces the old one.

We have modified the RRC approach by incorporating Simulated Annealing (SA) when determining whether to accept a new partial solution. Instead of always replacing the old partial solution with a superior one, the acceptance of the new partial solution is guided by the SA mechanism. This introduces a probabilistic component, allowing for the acceptance of potentially inferior solutions early in the process to escape local optima.

Simulated Annealing evaluates the new partial solution based on a temperature parameter that decreases over time. At higher temperatures, the method is more likely to accept worse solutions, fostering exploration, while at lower temperatures, it becomes more selective, emphasizing exploitation of the best solutions. This modification improves the robustness of RRC and enhances the quality of the final solution by effectively refining suboptimal segments.

6.1.6 Experiments

In this section, we compare our method with LEHD model. We report the optimality gap. The optimality gap measures the difference between solutions achieved by different learning methods and the (near) optimal solutions obtained using LKH3. Note that, as LKH3 is not an exact algorithm, it will give sub-optimal solutions. So, the optimzality gap can be negative too.

The main experimental results on uniformly distributed CVRP instances are reported in Table 6.1. We have obtained results for 50, 100 and 200

[2]`https://pan.baidu.com/s/12uxjol_5pAlnm0j4F6D_RQ?pwd=rzja`

	i50		i100		i200	
	rrc	rrc+sa	rrc	rrc+sa	rrc	rrc+sa
CVRP100	**0.535%**	0.558%	**0.272%**	0.284%	0.106%	**0.096%**
CVRP200	0.515%	**0.440%**	0.217%	**0.169%**	-0.032%	**-0.112%**
CVRP500	0.930%	**0.891%**	**0.546%**	0.596%	0.223%	**0.189%**
CVRP1000	2.814%	**2.705%**	2.370%	**2.252%**	**1.826%**	1.860%
Overall	**0.568%**	0.587%	0.300%	**0.283%**	0.127%	**0.117%**

Table 6.1: Experimental results with uniformly distributed instances. i50, i100, and i200 represent 50, 100, and 200 iterations, respectively.

iterations. The integration of Simulated Annealing has demonstrated better results in most cases.

6.2 Reinforcement Learning

6.2.1 Policy Gradient Framework

Reinforcement learning focuses on solving sequential decision-making problems, where an agent interacts with an environment to maximize cumulative rewards.Here, we outline the key concepts, starting with the definition of a policy, the associated objective function, and the procedure to update the policy.

Policy Definition

The policy π_θ is a probabilistic mapping from the current state and problem instance to an action. Formally, it is represented as:

$$\pi_\theta(a_{t+1} \mid \rho, \tau_t)$$

where:

- θ: Parameters of the policy that we aim to optimize.
- a_{t+1}: The action to be taken at step $t + 1$.
- ρ: The problem instance, sampled from a distribution of problem instances $\mathcal{D}$.

- $\tau_t = (x_1, \ldots, x_t)$: The solution instance, which includes all states traversed up to time t.

The policy π_θ represents the probability of taking action a_{t+1} at a given time t, given the current problem instance and the solution constructed so far. This probability is critical for exploring and exploiting potential solutions.

Probability of a Complete Solution Path

The probability of reaching any specific solution instance τ (a sequence of states and actions) is the product of the probabilities of taking all individual steps:

$$\pi_\theta(\tau) = \prod_t \pi_\theta(a_{t+1} \mid \rho, \tau_t).$$

This formulation ensures that the overall probability of a trajectory is decomposed into stepwise decisions, allowing the optimization of the entire solution path.

Objective Function

To train the policy, we define an objective function that quantifies the quality of the solution paths generated by the policy. The reward for a given solution instance τ is represented by $R(\tau)$. The objective is to maximize the expected reward over all possible trajectories and problem instances.

$$J(\theta) = \mathbb{E}_{\rho \sim \mathcal{D}} \left[\mathbb{E}_{\tau \sim \pi_{\theta,\rho}} \left[R(\tau) \right] \right].$$

Here:

- $R(\tau)$: The reward function evaluates how good a solution instance τ is.
- $\mathbb{E}_{\tau \sim \pi_{\theta,\rho}}$: The expectation over solution instances τ sampled according to the policy π_θ for a specific problem instance ρ.
- $\mathbb{E}_{\rho \sim \mathcal{D}}$: The expectation over the distribution of problem instances.

The goal is to optimize the policy parameters θ such that the expected reward $J(\theta)$ is maximized.

Policy Update

To maximize the objective function, we use gradient ascent. The gradient of $J(\theta)$ with respect to the policy parameters θ guides how the policy should be updated. The update rule is:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta),$$

where:

- α: The learning rate, which determines the step size of the updates.
- $\nabla_\theta J(\theta)$: The gradient of the objective function with respect to the policy parameters θ.

This update rule ensures that the policy parameters are adjusted in the direction that increases the expected reward $J(\theta)$. Over successive iterations, the policy becomes better at generating high-quality solution paths.

6.2.2 POMO - Policy Optimization with Multiple Optima

Motivation

In combinatorial optimization (CO) problems, we often deal with tasks such as finding the shortest path, solving the Traveling Salesperson Problem (TSP), or similar graph-based problems. Here, we outline the key challenges associated with standard reinforcement learning methods and introduce the motivation for using POMO (Policy Optimization with Multiple Optimas) to mitigate these issues.

Assume we are given a combinatorial optimization problem with a group of nodes $\{v_1, v_2, \ldots, v_M\}$ and a trainable policy network. The policy is tasked with generating a valid solution τ step-by-step. Formally, the policy π_t for selecting the next action at step t is defined as:

$$\pi_t = \begin{cases} p_\theta(a_t \mid s) & \text{for } t = 1, \\ p_\theta(a_t \mid s, a_{1:t-1}) & \text{for } t \in \{2, 3, \ldots, M\}, \end{cases}$$

where:

- a_t: The action (node) selected at time step t.

- s: The state defined by the problem instance.
- p_θ: The policy parameterized by θ.

This framework enables the generation of solutions by incrementally selecting actions based on the current state and past actions.

In many CO problems, a single solution can often be expressed in multiple equivalent forms due to symmetry. For example, in the case of the Traveling Salesperson Problem (TSP):

$$\tau = (v_1, v_2, v_3, v_4, v_5)$$

is equivalent to:

$$\tau' = (v_2, v_3, v_4, v_5, v_1),$$

since both represent the same closed tour. However, standard learning methods often fail to exploit this equivalence and become heavily biased by the starting point of the solution.

The starting action significantly influences the agent's future decisions, leading to asymmetry in the learning process. For example:

- If the agent starts at a specific node, it might consistently favor certain solution paths over others, even when they are equivalent.
- This bias reduces the diversity of explored solutions and limits the policy's ability to generalize across problem instances.

In order to solve this problem , POMO offers the following modifications:

- **Multiple Starting Points**
  POMO begins with designating N different nodes $\{a_1^1, a_2^1, \ldots, a_N^1\}$ as starting points for exploration (Figure 2, (b)). The network samples N solution trajectories $\{\tau_1, \tau_2, \ldots, \tau_N\}$ via Monte-Carlo method, where each trajectory is defined as a sequence $\tau_i = (a_i^1, a_i^2, \ldots, a_i^M)$ for $i = 1, 2, \ldots, N$.
  Previous methods use a fixed $START$ token , which induces bias based on the initial token. POMO reduces this bias by creating N solution trajectories from N different starting nodes.
  Conceptually, explorations by POMO are analogous to guiding a student to solve the same problem repeatedly from many different angles, exposing her to a variety of problem-solving techniques that would otherwise be unused.

- **A shared baseline for policy gradients**
  Once we sample a set of solution trajectories $\{\tau_1, \tau_2, \ldots, \tau_N\}$, we can calculate the return (or total reward) $R(\tau_i)$ of each solution τ_i. To maximize the expected return J, we use gradient ascent with an approximation

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (R(\tau_i) - b_i(s)) \nabla_\theta \log p_\theta(\tau_i|s)$$

  where $p_\theta(\tau_i|s) \equiv \prod_{t=2}^{M} p_\theta(a_t|s, a_{1:t-1})$.

  $b_i(s)$ is a baseline that one has some freedom of choice to reduce the variance of the sampled gradients. In principle, it can be a function of a_1, assigned differently for each trajectory τ_i. In POMO, however, we use the shared baseline,

$$b_i(s) = b_{shared}(s) = \frac{1}{N} \sum_{j=1}^{N} R(\tau_j) \quad \text{for all } i.$$

  The shared baseline used by POMO makes RL training highly resistant to local minima.With the shared baseline, each trajectory now competes with N 1 other trajectories where no two trajectories can be identical.

- **Multiple greedy trajectories for inference**
  Neural network models designed for combinatorial optimization (CO) problems generally offer two modes for inference. It can use the greedy method or the sampling method. While sampled solutions typically yield lower rewards on average compared to the greedy solution, repeated sampling can uncover solutions with higher rewards at the cost of increased computational effort.

  POMO introduces a novel approach by producing multiple deterministic greedy trajectories instead of relying on stochastic sampling. Starting from N different initial nodes $\{a_1^1, a_1^2, \ldots, a_1^N\}$, the model deterministically generates N distinct greedy trajectories. This allows for selecting the best solution among these trajectories, similar to the strategy

in sampling mode. However, the N greedy trajectories from POMO generally outperform N sampled trajectories, offering better solutions.

6.2.3 Experiments

Policy optimization involves selecting actions or decisions based on a trained policy during inference. Below are the techniques used for inference in this context, along with detailed descriptions:

Softmax

- The Softmax function converts raw output scores (logits) into a probability distribution over the possible next nodes.
- This allows probabilistic selection, with higher likelihood nodes receiving more weight.
- The formula for Softmax is:

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

  where:

  - y_i: Logit value of the i-th node.
  - n: Total number of nodes.

- This method maintains diversity in exploration by considering all nodes, weighted by their probabilities.

Greedy

- The Greedy method deterministically selects the node with the highest probability.
- It avoids randomness, directly choosing the node with the maximum value in the probability distribution.
- The next node is computed as:

$$\text{Next node} = \arg\max_i(P_i)$$

  where:

- P_i: Probability of the i-th node.

- This method is computationally efficient but may lead to suboptimal solutions by limiting exploration.

Gumbel-Softmax Sampling

- This technique adds controlled random noise to the logits using the Gumbel distribution, promoting exploration.

- The modified probabilities are then passed through a Softmax function for selection.

- The formula for Gumbel-Softmax is:

$$y_i = \frac{\exp\left(\frac{z_i + g_i}{\tau}\right)}{\sum_{j=1}^{k} \exp\left(\frac{z_j + g_j}{\tau}\right)}$$

  where:

  - z_i: Log-probability value for the i-th node.
  - g_i: Gumbel noise added to z_i.
  - τ: Temperature parameter controlling randomness.
  - k: Total number of nodes.

- Lower values of τ make the selection more deterministic, while higher values encourage exploration.

Epsilon-Greedy

RL exploration strategy where the agent chooses a random action with probability ϵ and the best-known action with probability $1 - \epsilon$.

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \text{best-known action}, & \text{with probability } 1 - \epsilon. \end{cases}$$

In summary, the results are shown in Table 6.2.

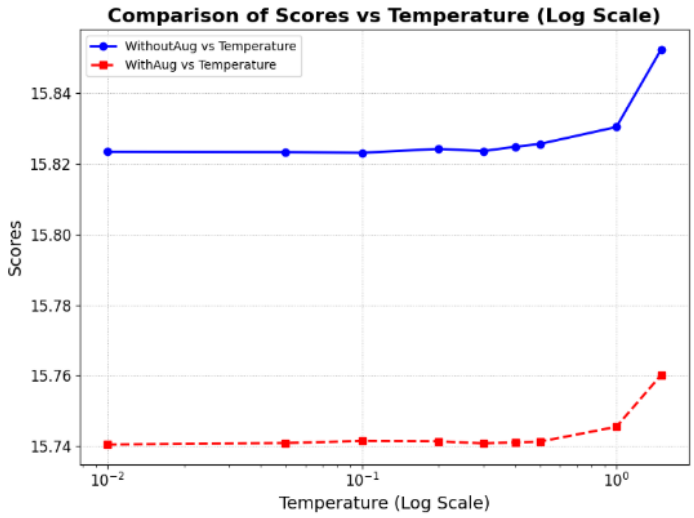

Figure 6.2: Experiments using different temperatures

Method	Without Augmentation	With Augmentation
Argmax	**15.8242**	15.7412
Softmax-Sampling	15.8368	15.7544
Epsilon-Greedy	15.8719	15.7695
Gumbel-Softmax	15.8234	**15.7405**

Table 6.2: Experiments using different Inference methods with and without augmentation

Instance Augmentation

One limitation of POMO's multi-greedy inference method lies in the fact that the number of greedy rollouts (N) it can perform is inherently constrained by the finite number of possible starting nodes. However, this limitation can be addressed by leveraging *instance augmentation*, a natural extension of POMO's core idea. The objective of instance augmentation is to find alternative ways of reaching the same optimal solution. For instance, the problem can be reformulated so that the model perceives a different configuration but ultimately produces the same result. An example of this is flipping or rotating the coordinates of all nodes in a 2D routing optimization task, thereby

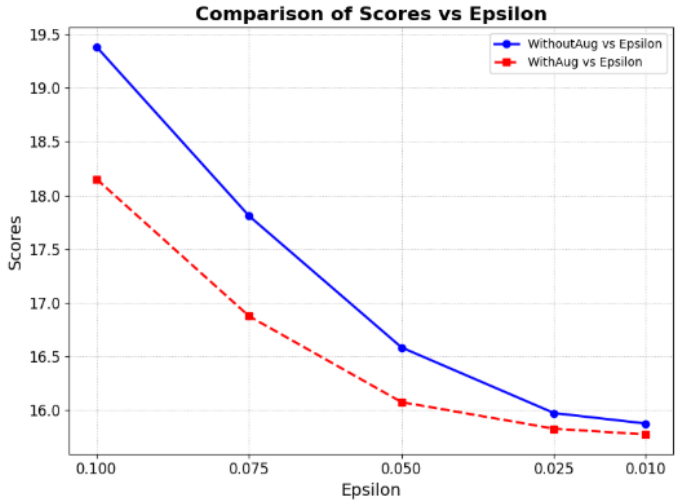

Figure 6.3: Experiments using different values of epsilon

creating new instances that allow for more diverse greedy rollouts.

In POMO, it uses 8 fold augmentation by default.

$$\begin{array}{cc} (x, y) & (1-x, y) \\ (x, 1-y) & (1-x, 1-y) \\ (x, y) & (x, -y) \\ (-x, y) & (-x, -y) \end{array}$$

In our experiment , we modified those augmentations , to be **2-fold** , **4-fold** and **8-fold augmentation with rotations**. The result of the experiments is shown in Table 6.3.

6.2.4 Adding Beam Search with POMO

Our final technique that we used is to encorporate Beam search with POMO. Code is available at: `https://github.com/wjalal/pomo_beam`.

- Beam search is a heuristic search algorithm that explores a graph by expanding only a limited set of the most promising nodes, modifying best-first search to reduce memory usage.
- It uses breadth-first search to build a search tree. At each level, it generates all successors, sorts them by heuristic cost, and keeps only a fixed number (β, the beam width) of best state.

Method	Augmentation 2 Fold	Augmentation 4 Fold	Augmentation 8 Fold (Horizontal and Vertical Flip)	Augmentation 8 Fold (Rotation)
Argmax	15.7885	15.7915	15.7412	15.7743
Softmax-Sampling	15.8164	15.7586	15.7544	15.7676
Epsilon-Greedy	15.7915	15.7908	15.7695	15.7992
Gumbel-Softmax	15.7786	15.7582	15.7405	15.7678
Beam Search	**15.7422**	**15.7288**	**15.6985**	**15.7412**

Table 6.3: Experiments using different Augmentation methods

Method	Without Augmentation	With Augmentation
Argmax	15.8242	15.7412
Softmax-Sampling	15.8368	15.7544
Epsilon-Greedy	15.8719	15.7695
Gumbel-Softmax	15.8234	15.7405
Beam Search	**15.7576**	**15.6985**

Table 6.4: Experiments using different Inference methods with and without augmentation

- Larger beam widths reduce pruning; infinite width makes it equivalent to best-first search. Beam width of 1 makes it identical to hill climbing.
- A goal state may be pruned, so it doesn't guarantee a solution if one exists, or the best solution. The beam width determines the memory required for the search.

In our approach , we maintain a beam_size to indicate how the search for new nodes are explored. We also keep track of the log probability of generated trajectories in order to find the current best sequence. From K initial states , beam search generates K^2 states. In the pruning state , we pick the top K states and continue.

Algorithm 1 Beam Search Forward

function FORWARD(state)
    $batch\_size \leftarrow$ state.BATCH_IDX.size(0)
    $pomo\_size \leftarrow$ state.BATCH_IDX.size(1)
    $beam\_size \leftarrow$ state.BATCH_IDX.size(2)
    $prob \leftarrow \mathbf{1}$ ▷ Initialize probabilities to ones of size $(batch\_size, pomo\_size, beam\_size)$
    $selected\_beams \leftarrow \text{arange}(beam\_size)$ ▷ Expand indices for all beams

    if $state.selected\_count = 0$ **then** ▷ First move (depot)
        $selected \leftarrow \mathbf{0}$ ▷ Zeros of size $(batch\_size, pomo\_size, beam\_size)$

    else if $state.selected\_count = 1$ **then** ▷ Second move (POMO)
        $selected \leftarrow \text{arange}(1, pomo\_size + 1)$ ▷ Expand sequential indices for $beam\_size$

    else if $state.selected\_count = 2$ **then** ▷ Third move (BEAM)
        $probs \leftarrow \text{Decoder}(state.current\_node[:, :, 0], state.load[:, :, 0], state.ninf\_mask[:, :, 0])$
        $(prob, selected) \leftarrow \text{TopK}(probs, beam\_size)$

    else ▷ Later moves
        for $k \leftarrow 0$ to $beam\_size - 1$ **do**
            $probs \leftarrow \text{Decoder}(state.current\_node[:, :, k], state.load[:, :, k], state.ninf\_mask[:, :, k])$
            $(selected_k, prob_k) \leftarrow \text{TopK}(probs, beam\_size)$
            $logprob_k \leftarrow state.logprob[:, :, k] + \log(prob_k + 10^{-5})$
        Concatenate results for *selected*, *prob*, and *logprob*
        Refine selections using top-k log probabilities
    return $selected, selected\_beams, prob$

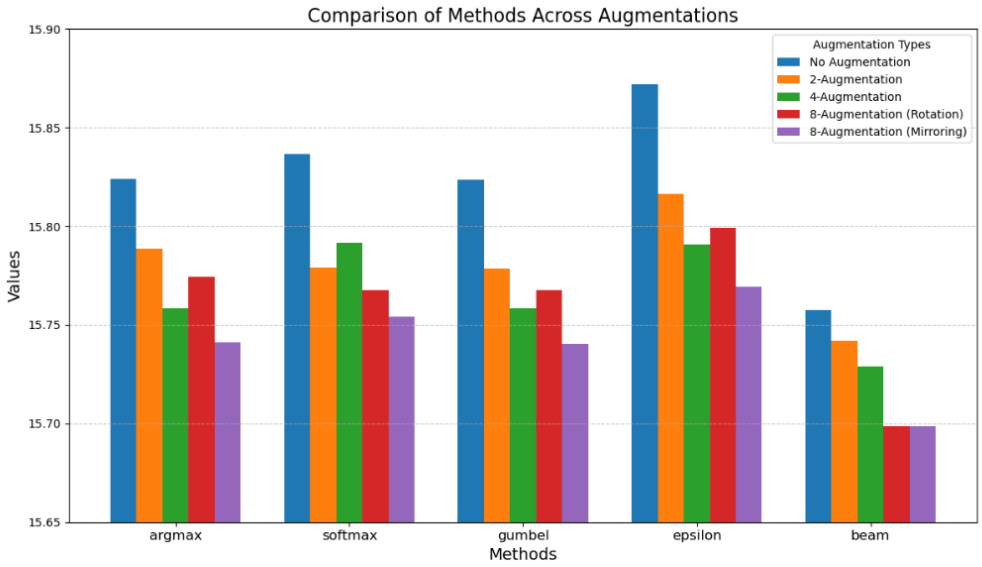

Figure 6.4: Comparison of different inference techniques and augmentation types.

Bibliography

[1] Asma M Altabeeb, Abdulqader M Mohsen, and Abdullatif Ghallab. An improved hybrid firefly algorithm for capacitated vehicle routing problem. *Applied Soft Computing*, 84:105728, 2019.

[2] Kemal Altinkemer and Bezalel Gavish. Heuristics for unequal weight delivery problems with a fixed error guarantee. *Operations Research Letters*, 6(4):149–158, 1987.

[3] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

[4] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283, 2011.

[5] Amariah Becker, Philip N Klein, and David Saulpic. A quasi-polynomial-time approximation scheme for vehicle routing on planar and bounded-genus graphs. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2017.

[6] Amariah Becker, Philip N Klein, and David Saulpic. Polynomial-time approximation schemes for k-center, k-median, and capacitated vehicle routing in bounded highway dimension. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[7] Amariah Becker, Philip N Klein, and Aaron Schild. A ptas for bounded-capacity vehicle routing in planar graphs. In *Algorithms and Data*

Structures: 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5–7, 2019, Proceedings 16, pages 99–111. Springer, 2019.

[8] Jannis Blauth, Vera Traub, and Jens Vygen. Improving the approximation ratio for capacitated vehicle routing. *Mathematical Programming*, 197(2):451–497, 2023.

[9] Vincent Cohen-Addad, Arnold Filtser, Philip N Klein, and Hung Le. On light spanners, low-treewidth embeddings and efficient traversing in minor-free graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 589–600. IEEE, 2020.

[10] Claudio Contardo. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. 2012.

[11] Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129–146, 2014.

[12] Raafat Elshaer and Hadeer Awad. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, 140:106242, 2020.

[13] Alfian Faiz, Subiyanto Subiyanto, and Ulfah Mediaty Arief. An efficient meta-heuristic algorithm for solving capacitated vehicle routing problem. *International Journal of Advances in Intelligent Informatics*, 4(3):212–225, 2018.

[14] Zachary Friggstad, Ramin Mousavi, Mirmahdi Rahgoshay, and Mohammad R Salavatipour. Improved approximations for capacitated vehicle routing with unsplittable client demands. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 251–261. Springer, 2022.

[15] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106:491–511, 2006.

[16] Mordecai Haimovich and Alexander HG Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of operations Research*, 10(4):527–542, 1985.

[17] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

[18] Ali Asghar Rahmani Hosseinabadi, Najmeh Sadat Hosseini Rostami, Maryam Kardgar, Seyedsaeid Mirkamali, and Ajith Abraham. A new efficient approach for solving the capacitated vehicle routing problem using the gravitational emulation local search algorithm. *Applied Mathematical Modelling*, 49:663–679, 2017.

[19] Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

[20] Hao Jiang, Mengxin Lu, Ye Tian, Jianfeng Qiu, and Xingyi Zhang. An evolutionary algorithm for solving capacitated vehicle routing problems by using local information. *Applied Soft Computing*, 117:108431, 2022.

[21] Michael Khachay and Roman Dubinin. Ptas for the euclidean capacitated vehicle routing problem in. In *International Conference on Discrete Optimization and Operations Research*, pages 193–205. Springer, 2016.

[22] Thau Soon Khoo and Babrdel Bonab Mohammad. The parallelization of a two-phase distributed hybrid ruin-and-recreate genetic algorithm for solving multi-objective vehicle routing problem with time windows. *Expert Systems with Applications*, 168:114408, 2021.

[23] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[24] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.

[25] Habibeh Nazif and Lai Soon Lee. Optimised crossover genetic algorithm for capacitated vehicle routing problem. *Applied Mathematical Modelling*, 36(5):2110–2117, 2012.

[26] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9:61–100, 2017.

[27] A Pessoa. Robust branch-cut-and-price algorithms for vehicle routing problems. *The Vehicle Routing Problem/springer*, 2008.

[28] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and Francois Vanderbeck. In-out separation and column generation stabilization by dual price smoothing. In *Experimental Algorithms: 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings 12*, pages 354–365. Springer, 2013.

[29] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & operations research*, 31(12):1985–2002, 2004.

[30] Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 2006.

[31] Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

[32] Stefan Røpke. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. *Presentation in Column Generation*, 2012, 2012.

[33] Mohammad Sajid, Jagendra Singh, Raza Abbas Haidri, Mukesh Prasad, Vijayakumar Varadarajan, Ketan Kotecha, and Deepak Garg. A novel algorithm for capacitated vehicle routing problem for smart cities. *Symmetry*, 13(10):1923, 2021.

[34] Ines Sbai, Saoussen Krichen, and Olfa Limam. Two meta-heuristics for solving the capacitated vehicle routing problem: the case of the tunisian post office. *Operational Research*, pages 1–43, 2022.

[35] Boon Ean Teoh, Sivalinga Govinda Ponnambalam, and Ganesan Kanagaraj. Differential evolution algorithm with local search for capacitated vehicle routing problem. *International Journal of Bio-Inspired Computation*, 7(5):321–342, 2015.

[36] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.