Universiteit Leiden
The Netherlands

Opleiding Informatica

Accelerating solving the Vehicle Routing Problem by using generative initialization

Niels Versteeg (s3322637)

Supervisors:
Dr. F. Ye
L.R. Arp MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

15/07/2025

Abstract

The Vehicle Routing Problem (VRP) is a generalization of the well-known Travelling Salesperson Problem (TSP). The problem seeks to determine the optimal set of routes that a fleet of vehicles must traverse to reach a given set of customers. In this study, we compare the performance of a classical algorithm and its combination with a machine learning model that provides an initial set of solutions to optimize and investigate if there exists a performance gap. We propose a hybrid method called the Hybrid Machine-Learned Genetic Algorithm (Hybrid-MLGA), which combines a variant of the Genetic Algorithm (GA) with an encoder-decoder Graph Convolutional Network (GCN). In addition to the Hybrid-MLGA, we present an open-source implementation of the encoder-decoder GCN model. We find that the Hybrid-MLGA outperforms the standard GA up to a certain iteration budget, beyond which both methods converge to nearly identical average solutions across all tested problem sets. We show that a performance gap exists between randomly initializing the GA and initializing it with generated solutions, the latter accelerating the convergence of the Hybrid-MLGA.

Contents

1 Introduction

Efficient transportation from one location to another remains one of the central logistic challenges in modern society. Applications range from the distribution of essential medical supplies to hospitals, restocking supermarkets, to optimizing passenger transport. As demand continues to grow, so does the need for robust and scalable solutions to such routing problems.

The Vehicle Routing Problem (VRP), a generalization of the classic Travelling Salesperson Problem (TSP), lies at the heart of this challenge. Originally introduced by Dantzig and Ramser [DR59], who developed the first algorithmic approach for optimizing petrol deliveries, the VRP seeks to determine the most cost-effective set of routes for a fleet of vehicles to serve a given set of customers.

Numerous variants of the VRP have been defined, such as the Capacitated VRP (CVRP), where each vehicle has a limited carrying capacity; the VRP with Time Windows (VRPTW), which imposes service time constraints at customer locations; and the Multi-Objective VRP (MOVRP), which balances multiple competing goals such as cost, time, or the number of vehicles to use. In this work, we focus on the CVRP, the foundational variant of the VRP.

A broad spectrum of solution methods has been developed to address VRP variants, ranging from exact algorithms and classical heuristics to advanced meta-heuristics and, more recently, learning-based approaches. Consequently, research interest in the field has risen in recent years [NDP+24], driven by the growing need for efficient logistics in increasingly dynamic and complex environments.

The first approach to solve the VRP was proposed in [DR59] by Dantzig and Ramser, which was able to obtain near-optimal solutions to efficiently route petrol deliveries. Later, Clark and Wright proposed an algorithm for obtaining near-optimal routes for a fleet of capacitated trucks [CW64] within a set of limitations. As in the algorithm proposed by Dantzig and Ramser, the vehicles originated from a single depot, servicing a set of customers. For an in-depth and comprehensive review of the past and current approaches to solving the VRP, we direct you to [JST08, TY21, BCC+21, ZGYT22, BMMD24, SPGM24].

1.1 Classic Approach

Due to the NP-hard [TV02] nature of the Vehicle Routing Problem, a significant amount of research has been dedicated to solving the VRP using (meta) heuristics, as these have been widely recognized as efficient approaches to hard optimization problems [BLS13]. For example methods based on the Genetic Algorithm (GA) [Hol73].

One such approach was presented in [BA03] for the classic VRP, where the authors implement both a pure GA and a hybrid GA, which implements neighbourhood search to help the GA converge faster. The promising results helped establish a precedent that GAs are a strong competitor for solving the VRP.

Another method was proposed in [ORH06], which aims to solve the VRPTW by representing it as a multi-objective problem, minimizing the number of vehicles and the distance travelled. The authors introduced a GA that makes use of the Pareto ranking technique and fairly considers both dimensions of the problem.

Another recent approach in [LLZ25] solved the VRPTW by introducing an improved GA. The authors propose a heuristic initialization algorithm to generate high-quality initial populations, and a local search operation to improve the performance of the GA. For a more in-depth review of the

GA, see [KCK21].

In addition to the Genetic Algorithm, other algorithms that took inspiration from nature exist, such as the Ant Colony Optimization (ACO) algorithm [DG97a, DBS06]. For example, [BM04] simulates the decision-making process of ant colonies foraging for food and modified the original ACO algorithm for the TSP [DG97b] to allow searching for multiple routes.

In [WCLY16], a novel approach to the ACO algorithm was introduced which modified the original ACO algorithm by allowing ants to visit the depot(s) more than once, until all customers were visited. For a review of other ACO algorithms and its real-world applications, we direct you to [DG99] and [RMLG07].

Genetic and ACO algorithms are two of the most common meta-heuristic algorithms, but other methods which solve the VRP exist in the literature, such as a hybrid approach using tabu search and simulated annealing proposed in [LLYL09], or the exact algorithm proposed in [AB17] which solved the G-VRP, a variant of the VRP that includes potential refuelling stops along a route. The authors model the G-VRP as a set partitioning problem where columns represent feasible routes. A taxonomic review of other (meta) heuristic algorithms is presented in [EA20].

1.2 Machine Learning Approach

In recent years, learning to solve the VRP using machine learning (ML) has garnered much attention. As shown in [SPGM24], the number of papers related to learning the VRP has almost doubled in the last four years. Many ML approaches have been proposed, such as the influential paper by Kool et al. [KvHW19] which introduced an attention-layer-based model that solved the CVRP, trained with the REINFORCE algorithm with a deterministic greedy rollout baseline.

Another method which uses attention is presented in [NOTS18]. The authors proposed an encoder-decoder model based on modified pointer networks [VFJ17]. They replaced the Recurrent Neural Network encoder and instead directly used the embedded inputs. Similarly to [KvHW19], the authors trained their model using the REINFORCE algorithm with a policy gradient approach.

1.3 Hybrid Methods

As discussed previously, much research has been done on solving the VRP using meta-heuristic algorithms and recently using ML methods. The question then arises: can these two methods be combined? Numerous research studies have been presented that do exactly this.

For example, in [QZW+24], the authors combined the encoder-decoder GCN-NPEC model presented in [DZH+20] with a GA to further optimize the solutions generated by the model. They trained the model with same methodology proposed in [DZH+20], using a REINFORCE algorithm with a rollout baseline.

Another method based on the encoder-decoder structure is presented in [WWH+24]. In their paper, the authors proposed a hybrid approach to solve the MOVRPTW (Multi-Objective VRP with Time Windows) that combined weight-aware deep reinforcement learning (WADRL) with the classical Non-dominated Sorting Genetic Algorithm II (NSGA-II) [DPAM02] to optimize the solution generated.

The preliminary work has shown that both machine learning and optimization methods yield promising results, and that integrating machine learning with traditional optimization methods has significant potential. However, both approaches face challenges related to complexity. Meta-heuristic

algorithms may require a substantial amount of time to reach satisfactory solutions, especially as instances increase in size, while relatively simple machine learning models show a significant performance gap to optimal solutions and struggle to generalize to unseen problem instances.

There exists a gap in performance between the two models: either a high quality solution is found at the cost of long computation times, or a faster method yields poor solutions. The literature suggests that combining meta-heuristic methods with learned models can improve performance [QZW+24], but it remains unclear how such hybrid approaches bridge this gap. This bachelor thesis explores how integrating a machine learning model with a classical algorithm influences the resulting performance. Specifically, our research question is:

What impact does providing initial solutions generated by a learned model to a metaheuristic algorithm have on its convergence speed and solution quality?

2 Problem Definition

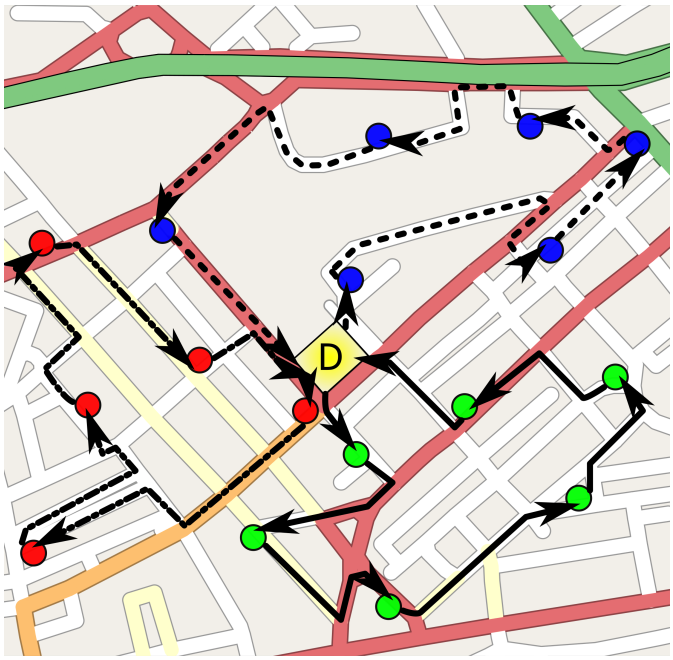


Figure 1: Example solution to the Vehicle Routing Problem using three vehicles. Adapted from *Vehicle Routing Problem Example* by Maly LOLek [LOL09].

In this section, we will define the problem of the classical VRP, also known as the Capacitated Vehicle Routing Problem (CVRP). An instance of the VRP can be defined on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V} = \{0, 1, \ldots, n\}$, where $i = 0$ is defined as the depot and the set $i \in \{1, 2, \ldots, n\}$ as the customer nodes. The set of edges can be defined as $\mathcal{E} = \{e_{i,j}\}$ with $i, j \in \mathcal{V}$. Each node is characterized by a two-dimensional characteristic vector $x_i = \{p_i, d_i\}$ where p_i is the position and d_i the demand of a node, with $x_0 = \{p_i\}$, and each edge by the distance between the connected nodes i and j, $w_{i,j}$. To generalize this problem to the real world, we take the graph $\mathcal{G}$ to be fully connected. In other words, we have an edge between each two nodes, where we define the edge $e_{i,j}$ different from $e_{j,i}$. The VRP employs a fleet of k vehicles, each with a capacity c_k.

The goal of the VRP is to find the set of optimal vehicle routes that a fleet of vehicles must traverse so that each customer is visited exactly once and each vehicle should start and end their route at the depot. An example solution to an instance of the VRP is shown in Figure 1.

3 Genetic Algorithm

Algorithm 1 Hybrid Genetic Search (HGS) for CVRP

1: **Input:** Initial set of solutions $\{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n\}$
2: **Output:** Best found solution $\boldsymbol{\pi}^*$
3: Initialize population $\mathcal{P} \leftarrow \{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n\}$
4: Let $\boldsymbol{\pi}^* \leftarrow \arg\min_{\boldsymbol{\pi} \in P} f(\boldsymbol{\pi})$
5: **repeat**
6: Select parents $(\boldsymbol{\pi}^{p_1}, \boldsymbol{\pi}^{p_2})$ from $\mathcal{P}$ via k-ary tournament
7: $\boldsymbol{\pi}^o \leftarrow \mathrm{XO}(\boldsymbol{\pi}^{p_1}, \boldsymbol{\pi}^{p_2})$ *Apply crossover*
8: $\boldsymbol{\pi}^c \leftarrow \mathrm{LS}(\boldsymbol{\pi}^o)$ *Improve offspring via search procedure*
9: Insert $\boldsymbol{\pi}^c$ into $\mathcal{P}$
10: **if** $f(\boldsymbol{\pi}^c) < f(\boldsymbol{\pi}^*)$ **then**
11: $\boldsymbol{\pi}^* \leftarrow \boldsymbol{\pi}^c$
12: **end if**
13: **if** $|\mathcal{P}| > \mu_{\max}$ **then**
14: Remove the least fit solutions until $|\mathcal{P}| = \mu_{\min}$
15: **end if**
16: **until** stopping criterion is met
return $\boldsymbol{\pi}^*$

For this research, we want to determine what impact providing a good initial solution has on the performance of a classical VRP solving algorithm. As discussed in the literature review, over the years many good approaches have been developed, and a common choice is a Genetic Algorithm. We choose a relatively new VRP solver package `PyVRP` [WLK24] that provides a high-performance implementation of the Hybrid Genetic Search (HGS) algorithm [VCGP13, Vid22], which is a variant of the Genetic Algorithm, specifically for vehicle routing problems.

In Algorithm 1, a pseudocode implementation of the HGS algorithm is shown. As mentioned before, the HGS is a variant of the Genetic Algorithm which combines the global search capabilities of a GA with local search methods. A GA works by evolving a population of solutions towards better solutions. Starting with a population which is initialized from a given list of (random) solutions, each iteration (generation) selects the k best individuals of the population and applies mutation and cross-over operators to obtain the next generation. This process is repeated until a maximum number of iterations is reached or the quality of the solution is satisfactory. Similarly to the GA, the HGS also maintains a population of solutions, initialized from a given list of solutions, as seen at Line 3. At Line 6, each generation selects two of the best existing solutions $\boldsymbol{\pi}^{p_1}$ and $\boldsymbol{\pi}^{p_2}$ from the parent population $\mathcal{P}$ using a k-ary tournament and applies the crossover operator XO at Line 7 to generate an offspring sequence that inherits features from both parent solutions. Unlike a regular GA, the HGS further improves the offspring by applying a local search procedure LS at

Line 8 and adds this improved solution to the current population at Line 9. When the population is full, as defined by $\mu_{\max}$, a survivor selection mechanism removes the least-fit solutions until the minimum population size $\mu_{\min}$ is reached at Line 14. This continues until the previously mentioned stopping criteria are met. In this study, the random initialization of the population at Line 3 is replaced with a set of solutions generated by a learned model.

4 Machine Learning Model

To address our research question, we require a relatively recent and effective model. From the literature, we adopt the Graph Convolutional Network with Node Sequential Prediction and Edge Classification (GCN-NPEC) [DZH+20], a relatively new and powerful approach. In this section, we outline the structure of the model, summarizing the key components as described by the original authors. Since certain details on the implementation of the model are not specified in the paper, we introduce several modifications, which will be discussed in the relevant sections.

4.1 Learning Goal

The GCN-NPEC model is built upon a Graph Convolutional Network (GCN) architecture, incorporating sequential node prediction and edge classification. Its goal is to generate a sequence of customers visits $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_T)$ in $t \in \{1, 2, \ldots, T\}$ steps, where $\pi_t \in \{0, 1, \ldots, n\}$. The depot (node 0) may be visited multiple times, with the customers in between each pair of depot visits defining a route. The main objective of the model as defined in [DZH+20] is:

$$\min c_v \cdot Q_v + c_t \sum_{t=1}^{T-1} w_{\pi_t, \pi_{t+1}} \tag{1}$$

Where c_v is the cost of a vehicle, Q_v the number of vehicles used and c_t the cost per unit of travel.

4.2 GCN-NPEC

The model is divided into three main components: the graph encoder, the sequential decoder, and the edge classifier. For a graphical overview of the model architecture, see the graphic provided in Figure 2. The encoder produces representations of the nodes and edges of the graph $\mathcal{G}$. The sequential decoder takes the encoded nodes as input and aims to generate a solution to a CVRP instance $\boldsymbol{\pi}$ in T steps, while the edge classifier aims to output a probability matrix that denotes the probability of an edge being present in a solution. The original authors define the sequence generated by the sequential decoder as the ground truth for the tours the edge classifier produces, creating a joint-learning approach.

4.2.1 Graph encoder

Given a CVRP instance graph $\mathcal{G}$, we want to encode the graph to extract its features. In [DZH+20], the authors split this into two main parts: the input encoding and the Graph Convolutional Network (GCN) [KW17] which extracts the graph features.

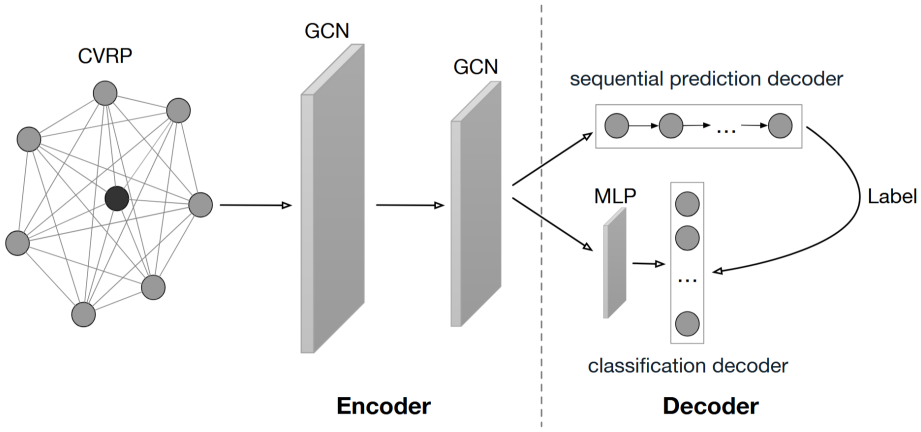


Figure 2: The architecture of the GCN-NPEC model. Figure reproduced from [DZH+20].

The input encoder encodes the nodes separately from the edges. As shown in Section 2, the features of the nodes x_i for $i \in \{1, 2, \ldots, n\}$ consist of their position and demand, and x_0 is given only by its coordinates. The node's features are initialized as vectors of dimension d_x, obtained by applying a fully connected (linear) neural network followed by the ReLU activation function:

$$x_i = \begin{cases} \text{ReLU}(W_1 x_i^p + b_1), & \text{if } i = 0 \\ \text{ReLU}(W_2 x_i^p + b_2; W_3 x_i^d + b_3), & \text{if } i > 0 \end{cases} \tag{2}$$

where W_1, W_2, W_3 are learnable weight matrices and b_1, b_2, b_3 are learnable parameters, and where $[\,\cdot\,;\,\cdot\,]$ represents the concatenation operator. Note that the dimension d_x only represents the size of the internal dimensions, as the concatenation operator changes the output dimension.

The edge features $y_{i,j}$ for $i, j \in \mathcal{V}$ consist of the edge length $w_{i,j}$ and the adjacency between the nodes i and j. Although $\mathcal{G}$ is fully connected, we define the adjacency matrix $A \in \mathbb{R}^{(n+1)\times(n+1)}$ as follows:

$$a_{i,j} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ } k\text{-nearest neighbours,} \\ -1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Thus, nodes i and j are adjacent if they are k-nearest neighbours. In [DZH+20] $k = 10$ is used. Finally, we use the length and adjacency of the edges to calculate the d_y dimensional edge features:

$$y_{i,j} = \text{ReLU}\left([W_4 w_{i,j} + b_4; W_5 a_{i,j} + b_5]\right) \tag{4}$$

where W_4, W_5 and b_4, b_5 are trainable weight matrices and parameters respectively.

4.2.2 Graph Convolutional Network

The learned node and edge features are further processed with a (modified) Graph Convolutional Network. The traditional GCN is an extension of the Graph Neural Network (GNN) [SGT+09]. A generic GNN may implement three types of layers [BBCV21]:

1. Permutation equivariant layers, which compute the node-wise features.

2. Local pooling layers, which coarsen the graph.

3. A permutation-invariant global pooling layer, which functions as the read-out layer.

The GCN extends the GNN by introducing a specific method to aggregate and update node features with localized or spatial convolution operators, which enable the GCN to better capture features, enabling more efficient training on large graphs.

In [DZH+20], the authors apply learned linear projections to the node and edge feature vectors to obtain d_h-dimensional embeddings. These embeddings serve as input to the first layer of the GCN and are defined as:

$$h_i^0 = W_{E_1} x_i + b_{E_1} \tag{5}$$

$$h_{e_{i,j}}^0 = W_{E_2} x_{e_{i,j}} + b_{E_2} \tag{6}$$

The GCN updates these embeddings through L layers, where each layer $\ell = \{1, 2, \ldots, L\}$ consists of two sublayers: aggregation and combination [XHLJ19]. The aggregation sublayer is defined as [XLT+18, XHLJ19]:

$$h_{N(i)}^\ell = \sigma\left(W^\ell \cdot \text{AGGREGATE}\left(\{h_u^{(\ell-1)}, \forall u \in N(i)\}\right)\right) \tag{7}$$

where $h_{N(i)}^l$ denotes the aggregated embeddings of the neighbours of node i; W^ℓ is a learnable weight matrix shared across all nodes at layer ℓ; σ is a non-linear activation function, such as the ReLU; $N(i)$ represents the neighbourhood of node i; and AGGREGATE is the function used to combine the embeddings of all nodes in $N(i)$.

The combination sublayer is then defined as follows [XLT+18, XHLJ19]:

$$h_i^\ell = \text{COMBINE}\left(h_i^{\ell-1}, h_{N(i)}^\ell\right) \tag{8}$$

Where COMBINE is a function that combines the node embeddings from the previous layer with the aggregated embeddings of its neighbourhood.

4.2.3 Tailored GCN Encoder

In [DZH+20], the authors extend the standard GCN to incorporate edge embeddings, allowing both node and edge features to be updated embeddings simultaneously. The updated node embeddings at layer ℓ are given by:

$$h_{N(i)}^\ell = \sigma\left(W_I^\ell \cdot \text{AGGREGATE}_I^\ell\left(\text{ATTENTION}\left(h_i^{\ell-1}, \{h_u^{\ell-1}, \forall u \ \in N(i)\}\right)\right)\right) \tag{9}$$

where W_I^l is a trainable weight matrix at layer ℓ, shared across all nodes; AGGREGATE_I is the aggregating function as defined earlier; ATTENTION is a function $f : h_{key} \times H_{values} \rightarrow h_{values}$ that maps a key feature vector ($h_i^{\ell-1}$) and a set of candidate feature vectors to a weighted sum of elements-i.e., the attention values [DZH+20]. These values are calculated using scaled dot-product attention [VSP+23]. Note that the attention function described in Equation (9) only takes two inputs, while the cited scaled dot-product attention takes three: queries, keys, and values. In our implementation, we compute the attention values as follows:

$$\text{ATTENTION}\left(h_i^{\ell-1}, h_j^{\ell-1}\right) = \text{softmax}\left(\frac{h_i^{\ell-1} \cdot h_j^{\ell-1}}{\sqrt{d_h}}\right) \cdot h_j^{\ell-1},\ \forall j\ \in N(i),\ i \in \mathcal{V} \tag{10}$$

In this equation, we treat both the keys and values as the neighbouring node embeddings $h_i^{\ell-1}$, as the goal is to determine the relative importance of each node $j \in N(i)$ with respect to the query node i.

For each edge $e_{i,j} \in \mathcal{E}$, we define its neighbourhood as the two nodes (i and j) it connects. We define the edge neighbourhood embeddings $h^l_{N(e_{i,j})}$ as follows:

$$h^\ell_{N(e_{i,j})} = \sigma\left(W_E^\ell \cdot \text{AGGREGATE}_E^\ell\left(\left\{h_{e_{i,j}}^{\ell-1}, h_i^{\ell-1}, h_j^{\ell-1}\right\}\right)\right) \tag{11}$$

$$\text{AGGREGATE}_E^\ell\left(\left\{h_{e_{i,j}}^{\ell-1}, h_i^{\ell-1}, h_j^{\ell-1}\right\}\right) = W_{e_1}^\ell h_{e_{i,j}}^\ell + W_{e_2}^\ell h_i^\ell + W_{e_3}^\ell h_j^\ell \tag{12}$$

where $W_E^\ell, W_{e_1}^\ell, W_{e_2}^\ell$ and $W_{e_3}^\ell$ are trainable weight matrices. This extended aggregation sublayer integrates both the node and edge embeddings. The extended combination layer is then defined as follows:

$$h_i^\ell = \left[V_I^\ell \cdot h_i^{\ell-1}; h_{N(i)}^\ell\right] \tag{13}$$

$$h_{e_{i,j}}^\ell = \left[V_E^\ell \cdot h_{e_{i,j}}^{\ell-1}; h_{N(e_{i,j})}^\ell\right] \tag{14}$$

where V_I^ℓ and V_E^ℓ are trainable weight matrices. The updated combination sublayer follows the strategy introduced in [HYL17], in which the learned node and edge embeddings are concatenated with their respective neighbourhood embeddings. As noted in [XLT+18], these combination sublayers function as a skip connections between layers. To expand on this, we additionally incorporate regular skip connections [HZRS16] and apply layer normalization [BKH16].

4.2.4 Two Decoders

To decode both the encoded nodes and edges, the authors of [DZH+20] propose a dual decoder with separate networks to decode both features independently. Following this approach, we implement a sequential prediction decoder and a classification decoder, according to [DZH+20].

Sequential decoder: Following the authors, we employ a Recurrent Neural Network (RNN) with a Gated Recurrent Unit (GRU) [CvMG+14], combined with a context-based attention mechanism known as the pointer network [VFJ17]. The decoder maps the encoded node embeddings to a sequence of nodes $\boldsymbol{\pi}$. We aim to generate this sequence in T steps, such that $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_T\}$, where $T \geq n+1$, as the depot can be visited multiple times and at least twice for each valid route. We denote the sequence up to a step t by π_t with $t \in \{1, 2, \ldots, T\}$. Following [DZH+20], our goal is to learn a stochastic policy $p(\boldsymbol{\pi} \mid S; \theta)$ that generates a solution sequence $\boldsymbol{\pi}$ on input S, while minimizing the objective defined in Equation (1). From [DZH+20], we obtain:

$$\begin{aligned} P\left(\boldsymbol{\pi} \mid S; \theta\right) &= \prod_{t=0}^{T-1} p(\pi_{t+1} \mid S, \pi_t; \theta) \\ &= \prod_{t=0}^{T-1} p(\pi_{t+1} \mid f\left(S, \theta_e\right), \pi_t; \theta_d) \end{aligned} \tag{15}$$

where $P(\boldsymbol{\pi} \mid S; \theta)$ denotes the total probability of generating the sequence $\boldsymbol{\pi}$ on an input S. This probability can be broken down into the product of probabilities for selecting each next node π_{t+1}, chosen on the input S, and previously chosen nodes π_t. The function $f(S, \theta_e)$ represents the graph encoder, parametrized by θ_e and θ_d the learnable parameters of the sequential decoder. Following standard practice in sequence modelling [SVL14, VFJ17], we use the GRU to encode the history of the generated sequence π_{t-1} into a fixed-length hidden state vector z_t, such that:

$$p(\pi_t \mid f(S, \theta_e), \pi_{t-1}; \theta_d) \approx p(\pi_t \mid f(S, \theta_e), z_t; \theta_d). \tag{16}$$

To implement $p(\pi_t \mid f(S, \theta_e), z_t; \theta_d)$, we use the previously described pointer network. The authors simplified the context weights by reducing the two learnable weight matrices to just one:

$$W_1 x + W_2 y = W_3[x; y] \tag{17}$$

where W_1, W_2, W_3 are learnable weight matrices. Let h_i^L be the node embeddings and z_t the hidden state of the final GRU layer. We then define the attention score weight of node i at step t $u_{t,i}$ as:

$$u_{t,i} = \begin{cases} -\infty, & \forall j \in N' \\ W_{h_d}^T \tanh(W_s[h_i^L; z_t]), & \text{otherwise,} \end{cases} \tag{18}$$

where $W_{h_d}^T$ and W_s are learnable weights. To avoid selecting unfeasible nodes, we apply a mask by assigning $-\infty$ to all nodes $j \in N'$, where N' denotes the set of infeasible nodes.

During decoding, a sequence $\boldsymbol{\pi}$ should adhere to the following rules to be considered a valid solution to the CVRP. Each vehicle has capacity c, with $c > 0$, and each node i has a demand $0 \leq d_i \leq c$, with $d_0 = 0$. At each step t we visit a node i and update its demand so that $d_i = 0$ and $c_t = c_{t-1} - d_i$, where $c_{t-1} - d_i \geq 0$.

A complete tour $\boldsymbol{\pi}$ contains at least one route $\boldsymbol{\pi}^{r_j}$, which starts and ends at the depot, and visits at least one customer node. The total demand served on a route $\boldsymbol{\pi}^{r_j}$ must not exceed the capacity of a vehicle, such that $\sum_{i \in \boldsymbol{\pi}^{r_j}} d_i \leq c$.

Because each route must begin and end at the depot and include at least one customer node, the depot cannot be selected in two consecutive decoding steps. Additionally, a node is considered infeasible if it has a remaining demand of $d_i = 0$, or if $d_i > c_t$. Thus, we define the set of masked nodes as:

$$N'_t = \begin{cases} N'_{t-1} \cup \{0\}, & \pi_{t-1} = 0 \text{ or } t = 0 \\ \{i \in \mathcal{V} \mid d_i = 0 \ \vee \ d_i > c_t\}, & \text{otherwise} \end{cases} \tag{19}$$

Finally, the masked attention scores are transformed into a pointing probability distribution towards the input nodes $p(\pi_t \mid f(S, \theta_e), \pi_{t-1}; \theta_d) = \text{softmax}(u_{i,t})$ for $i \in \mathcal{V}$ [VFJ17].

Classification decoder: While the sequential decoder generates a solution by selecting what customer to visit using pointing attention scores, the classification decoder predicts the likelihood of edges appearing in a solution. Specifically, it learns the edge-probability matrix $P_{\mathcal{E}} \in [0, 1]^{n \times n}$, where each entry $P_{\mathcal{E}_{i,j}}$ represents the probability that the edge between node i and j is included in the solution. A sequence $\boldsymbol{\pi}$ can then be constructed by traversing the edges indicated by $P_{\mathcal{E}}$. For example, consider the following edge-probability matrix:

$$P_{\mathcal{E}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Converting this matrix to a solution sequence gives $\boldsymbol{\pi} = (0, 1, 5, 2, 0, 3, 4, 0)$.

To generate the probability matrix, we apply a Multi-Layer Perceptron (MLP) on the encoded edge embeddings $h^L_{e_{i,j}}$, and convert the generated logits to a softmax distribution:

$$P_{\mathcal{E}} = \text{softmax}(\text{MLP}(h^L_{e_{i,j}})), \forall e_{i,j} \in \mathcal{E} \tag{20}$$

Combining both decoders, we implement a decoding loop that iteratively constructs a solution $\boldsymbol{\pi}$ over T steps. At each step $t \in \{1, 2, \ldots, T\}$, the next customer to visit is selected from the set of unmasked nodes $\mathcal{N} \setminus N'$, according to the chosen sampling strategy, thereby extending the previous solution π_{t-1}. This process is repeated until all n customers are visited, using $1 \leq k \leq n$ vehicles. A pseudocode outline of this procedure is provided in Algorithm 2.

Algorithm 2 Decoding Procedure

```
Initialize route π ← []
Initialize vehicle capacity C ← C_max
Initialize node mask N'_t based on initial constraints seen in Equation (19)
while not all customers served do
    Calculate the attention scores u_{t,i}
    if Greedy strategy then
        Select next node i_t ← arg max_i u_{t,i}
    else
        Sample next node i_t ∈ V from u_{t,i}
    end if
    Append i_t to route: π ← π ∪ {i_t}
    Update vehicle capacity: C ← C − x^d_{i_t}
    Set demand of i_t to zero: x^d_{i_t} ← 0
    Update mask N'_{t+1} based on new state
end while
return route π
```

4.2.5 Learning the VRP

Since obtaining optimal solutions for large VRP instances is often unfeasible, the authors propose a self-supervised training approach. Because the extended GCN-NPEC model simultaneously generates the node and edge embeddings, the solutions produced by the sequential and classification decoder must represent the same graph information and be consistent with each other. This principle forms the basis of the Joint-Learning Strategy proposed in [DZH+20].

The model is trained using a hybrid approach combining reinforcement learning (RL) and supervised learning (SL). Following the original work, we employ the same REINFORCE algorithm [SB98] with a rollout baseline. We calculate the advantages as the difference in the solution quality-number of vehicles used and total tour length-between a sampled decoder and a greedy decoder. We multiply the advantages by the sum of log probabilities of the selected nodes obtained during decoding. Unlike to the original paper, we normalize the advantages using PyTorch's `torch.normalize` function[1] to be in the range [...] to vastly improve model training convergence.

In parallel, we supervise the classification decoder by comparing the edge probability matrix $P_{\mathcal{E}}$ with the binary edge-probability matrix $P_{\mathcal{E}}^{\boldsymbol{\pi}} \in \{0,1\}^{n \times n}$, constructed from the sequence $\boldsymbol{\pi}$ generated by the sequential decoder. This is done by minimizing the negative sum of the cross-entropy losses between $P_{\mathcal{E}}$ and $P_{\mathcal{E}}^{\boldsymbol{\pi}}$. Finally, we combine both losses into a weighted loss function:

$$\mathcal{L}_\theta = \alpha \times \mathcal{L}_s(\theta) + \beta \times \mathcal{L}_r(\theta) \tag{21}$$

Where θ denotes the vector of all trainable parameters, and α, β the weighing coefficients for the reinforced and supervised loss.

4.3 Implementation

Since the authors of [DZH+20] did not provide source code, we implemented the model from scratch using the widely adopted machine learning framework PyTorch 2.7.0 [AYH+24] and the PyTorch Geometric (PyG) library [FL19]. Our implementation is divided into four main parts: Loading a VRP instance, encoding and decoding an instance, and generating a final solution. As part of this work, we provide an open-source implementation of the GCN-NPEC model, including training and evaluating tools, at `https://github.com/Plantius/vrp-gen-init`. Further details on the implementation are provided in Appendix A.

4.4 Training

We train the model largely following the original paper, with several modifications. We employ the Adam optimizer and reduce the learning rate by a factor of 0.96 every epoch, down to a lower limit of $3 \cdot 10^{-6}$. The cost per unit of travel is fixed at 1, and, contrary to the training process in the original paper, set the unit cost of a vehicle at 1 to discourage the use of more vehicles than the baseline. In Table 1, we provide an overview of the hyperparameter values used during training.

Due to computational and time constraints, we train three distinct versions of the model on datasets of sizes 20, 50 and 100 nodes. We generate a training set of $10,000$ instances for each size, using the generator script provided in [QSUV22]. All instances are created by sampling from a random mixture of available parameter configurations (see Table 2). We train each model for 1000 epochs, with mini-batch sizes adapted according to the dataset dimension.

5 Experiment

To evaluate our research question, we compare the performance of a Genetic Algorithm and its combination with the GCN-NPEC model. For the GA, we select a recent open-source implementation

[1] `https://docs.pytorch.org/docs/stable/generated/torch.nn.functional.normalize.html`

Component	Hyperparameter Setting
α, β	1.0
Optimizer	Adam
Learning Rate	Initial: $1 \cdot 10^{-3}$, decayed by 0.96 per epoch
Learning Rate Floor	$3 \cdot 10^{-6}$
Gradient Clipping	L2 norm clipped to 5.0
Weight Initialization	Uniform in $[-0.08, 0.08]$, bias set to 0
Decoder GRU	2 layers, 256 hidden units
Encoder GCN	3 layers
Classification Decoder MLP	3 layers

Table 1: Hyperparameters used during training of the model components.

of the Hybrid Genetic Search (HGS) algorithm (`PyVRP` [WLK24]) that builds upon the foundational work of [VCGP13]. The goal of this experiment is to assess whether providing an initial (set of) solution(s) to the HGS algorithm can:

1. Improve the quality of the solution
2. Accelerate the convergence time

We will evaluate the relative performance of the combine HGS algorithms, referred to as the Hybrid Machine Learned Genetic Algorithm (Hybrid-MLGA), against its standard counterpart, referred to as the baseline, on multiple test sets of varying problem sizes. Additionally, we compare different variants of the Hybrid-MLGA against each other. To test the absolute performance of the baseline and hybrid algorithms, we will evaluate their performance on the XML-100[2] benchmark set [QSUV22], and compare their results with the optimal solution.

5.1 Datasets

We generate five datasets named VRP-TEST-20, VRP-TEST-50, VRP-TEST-100, VRP-TEST-200, and VRP-TEST-400, where the appended number indicates the dimensionality of the problem set. Each test set contains 100 instances generated using the generator script provided by [QSUV22], sampling from a random mixture of available options. Note that individual instances within each problem set may specify different vehicle fleet capacities. In addition to the generated problem sets, we also evaluate the performance of the algorithms on the XML-100 benchmark set, which consists of $10,000$ instances of 100 nodes, for which optimal solutions are provided.

5.2 Setup

Our experiment is divided into three main parts: (1) comparing the performance of the Hybrid-MLGA with the standard HGS algorithm, (2) comparing the Hybrid-MLGA variants against each other, and (3) assessing the absolute performance of the Hybrid-MLGAs and the baseline algorithm. We tested four variants of the model with greedy decoding:

[2] http://vrp.atd-lab.inf.puc-rio.br/index.php/en/new-instances

Instance Feature	**Options**
Depot positioning	1. Random 2. Centred 3. Cornered
Customer positioning	1. Random 2. Clustered 3. Random-Clustered
Demand distribution	1. Unitary 2. Small values with large CV (Coefficient of Variation) 3. Small values with small CV 4. Large values with large CV 5. Large values with small CV 6. Depending on quadrant 7. Many small values and few large values
Average route size r where $r \subseteq \mathcal{N}$ and $\lvert r \rvert \leq \lvert \mathcal{N} \rvert$	1. Very short, $\lvert r \rvert \sim \mathcal{U}([3, 5])$ 2. Short, $\lvert r \rvert \sim \mathcal{U}([5, 8])$ 3. Medium, $\lvert r \rvert \sim \mathcal{U}([8, 12])$ 4. Long, $\lvert r \rvert \sim \mathcal{U}([12, 16])$ 5. Very long, $\lvert r \rvert \sim \mathcal{U}([16, 25])$ 6. Ultra long, $\lvert r \rvert \sim \mathcal{U}([25, 50])$

Table 2: Options used to generate the train and test problem sets, based on [QSUV22].

1. Model trained on VRP-TRAIN-20.
2. Model trained on VRP-TRAIN-50.
3. Model trained on VRP-TRAIN-100.
4. Randomly initialized model.

Each model will generates a set of one ore more initial solutions to initialize the population of the HGS algorithm. For this research, we did not apply hyperparameter optimization and left the parameters as default provided by the authors of `PyVRP`. For an overview of these parameters, see Table 3. The HGS algorithm is run for a maximum number of iterations $i = 250$ and the population is initialized with a set of solutions of size $j = 25$, matching the default minimum population size. We compare the absolute performance of the algorithms by comparing the optimal solutions of the XML-100 benchmark set with the best solution found across iterations.

Finally, to investigate whether the solutions generated by the Hybrid-MLGA and the baseline algorithm are statistically different, we apply a paired T-test on the total tour lengths of their best solutions found across all iterations and test instances.

Parameter	Value
Population	min size: 25, max size: 25 + 40 num elite: 4, num close: 5 lower bound diversity: 0.1, upper bound diversity: 0.5
Repair probability	0.8
Crossover operator	selective route exchange
Search method	local search

Table 3: Hyperparameters of the Hybrid Genetic Search algorithm.

6 Results & Conclusions

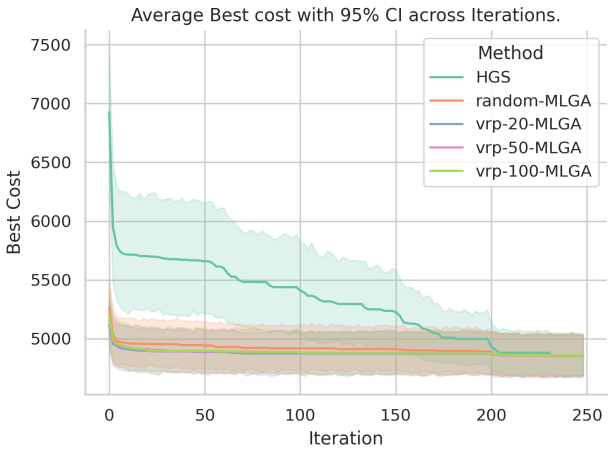
We divide our results and our conclusions into several subsections. We conducted our experiments using four models: one untrained model (randomly initialized) and the three models trained on the VRP-TRAIN-20, VRP-TRAIN-50, and VRP-TRAIN-100 datasets, respectively. The Hybrid-MLGAs were evaluated on five test sets containing 20, 50, 100, 200, and 400 customers. Each experiment was repeated five times, and all algorithms were run for a maximum of 250 iterations.

6.1 Hybrid-MLGA Converges Faster

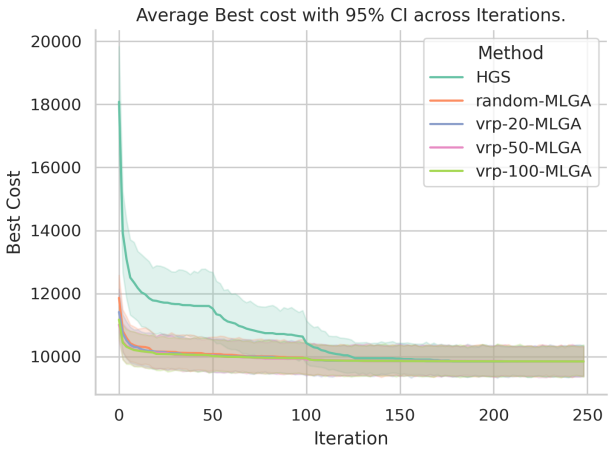
To investigate the difference between the randomly initialized HGS algorithm and the Hybrid-MLGA, we compare the average best cost found across iterations, shown in Figure 3. Each subfigure corresponds to one of the five problem sets and shows the average best cost with the 95% confidence interval (CI).

6.1.1 Performance Gap

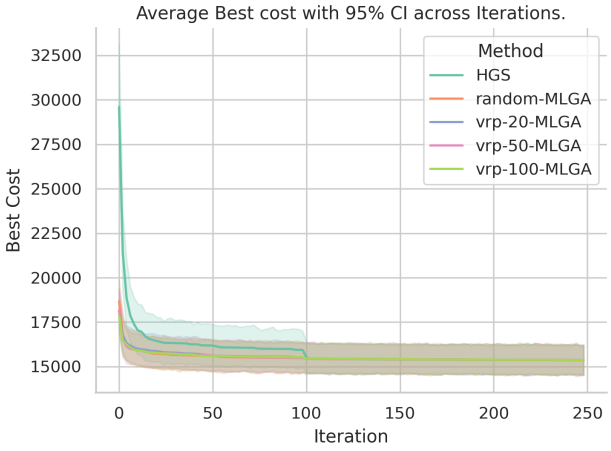
In Figure 3a the results for the VRP-TEST-20 set are shown. We observe that all four variants of the Hybrid-MLGA outperformed the standard HGS during the first ~ 200 iterations, after which all methods converged to a similar average solution quality. The random-GA performed slightly worse than the other Hybrid-GAs for the first ~ 200 iterations, and the model trained on 20 node instances provided the best initial solution. Similar to the results presented in Figure 3a, the results in Figure 3b indicate that the HGS performed worse than all Hybrid-MLGAs for the first 150 iterations. The random-MLGA again performed slightly worse, and the model trained on instances of size 50 provided the best initial solutions. The results in Figure 3c show that the HGS and Hybrid-MLGAs converged to a similar average best solution after ~ 100 iterations, and the initial best performer is the VRP-TRAIN-100-model. The results of the final two datasets are provided in Figure 3d and Figure 3e. Consistent with the previous results, Figure 3d shows that the HGS algorithm converged slower during the first ~ 100 iterations, and similarly in Figure 3e during the first ~ 50 iterations. In both cases, the baseline eventually reached a similar average solution as the MLGA variants. For both problem sets, the VRP-TRAIN-100-model, trained on the largest and thus most similar instance size to the test sets, initially provided the best solution. Based on our observations, we conclude that the HGS algorithm converges slower compared to all Hybrid-MLGA variants.



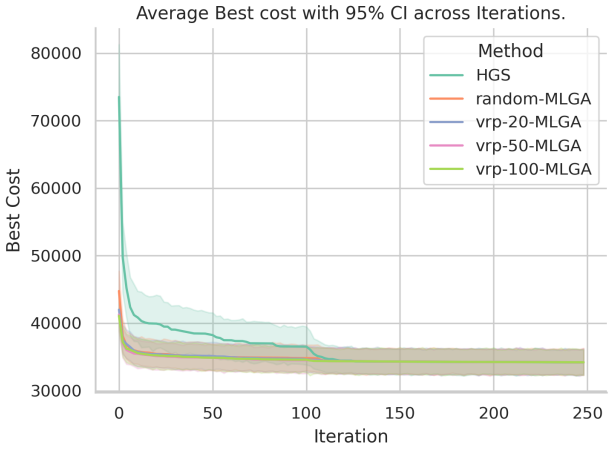
(a) VRP-TEST-20



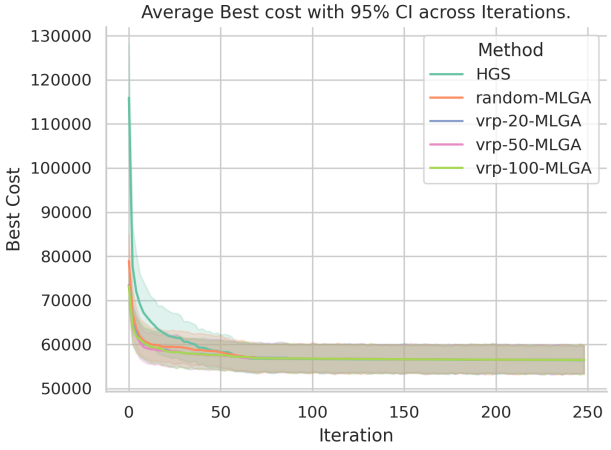
(b) VRP-TEST-50



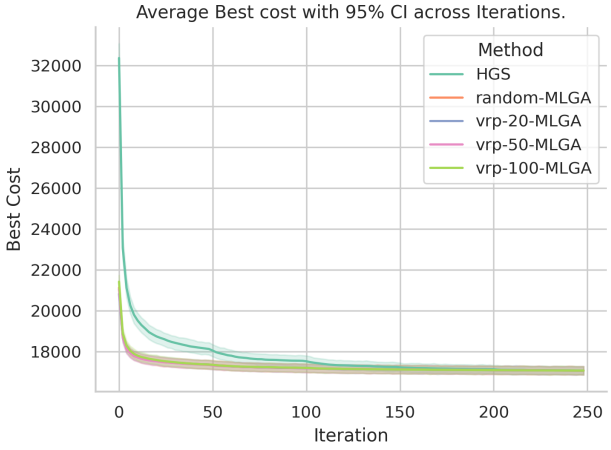
(c) VRP-TEST-100



(d) VRP-TEST-200



(e) VRP-TEST-400



(f) XML-100

Figure 3: The average best cost per iteration, shown as the mean line with 95% CI for each problem size. The prefix denotes the model used to initialize the population of the HGS.

In addition to the results presented in Figure 3, we provide a more detailed analysis in the form of five tables: Tables 5 to 9. These tables present a detailed overview of the best average solutions obtained by the five algorithms at selected iterations, along with the relative difference to the HGS algorithm for each Hybrid-MLGA variant.

6.1.2 Statistical Significance

To further support the validity of our results, we provide the statistical significance of the performance differences between the Hybrid-GAs and the baseline at each iteration, as shown in Figure 4. Each subfigure presents the p-value computed across all instances at a given iteration, with performance differences considered significant when $p < 0.05$.

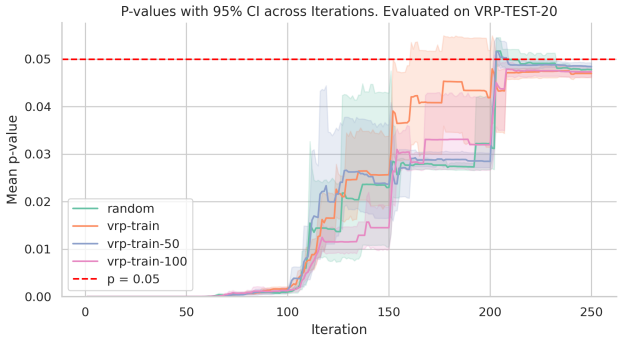
In Figure 4a, significant differences are observed for the first $\sim$ 150 iterations, aligning with the findings in Figure 3a, where the baseline and Hybrid-GA converge to similar average solutions (within their 95% CI) after approximately 150 iterations. A similar trend is seen in Figure 4b, where significant differences persist for the first $\sim$ 70 iterations, after which the 95% confidence interval of the baseline begins to overlap with the mean performance of the Hybrid-GAs. The remaining subfigures follow a comparable pattern. In Figure 4c the difference in performance is statistically significant for up to $\sim$ 50 iterations, with the baseline converging after approximately 100 iterations, although the CI overlap begins at iteration $\sim$ 10. Figure 4d shows statistically significant differences during the first $\sim$ 100 iterations, with the HGS algorithm converging around the same point. In Figure 4e, the significant differences are observed during the first $\sim$ 50 iterations and again after $\sim$ 170 iterations, while the HGS converges after approximately 50 iterations. From these observations we can conclude that the gap between the Hybrid-MLGA variants and the baseline algorithm is statistically significant.

6.1.3 Absolute Performance

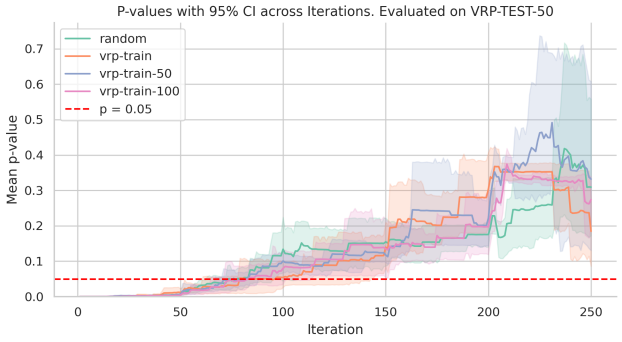
To compare the absolute performance of the five algorithms, we provide the average solution cost across iterations for the XML-100 benchmark set in Figure 3f. Furthermore, Table 4 presents the average solution cost for selected instances, along with the corresponding average gap to the optimal solution at selected iterations. As shown in Figure 3f, the HGS algorithm converged to a similar average solution after approximately 200 iterations. This is further confirmed in Table 4, where all five algorithms achieved an identical average optimality gap after around 200 iterations. Table 4 further shows that the Hybrid-GAs start with a substantially smaller optimality gap compared to the baseline algorithm. After 250 iterations, all methods converged to near optimal solutions, with final average gaps of 0.5% above the optimum. These observations indicate that the HGS algorithm reaches near-optimal solutions within the chosen budget of 250 iterations, suggesting that this iteration limit is appropriately selected.

6.1.4 Faster Convergence

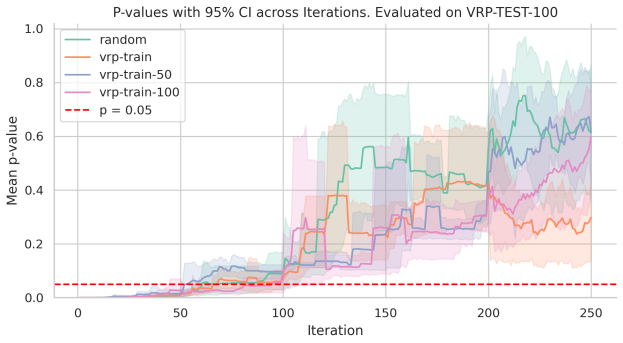
To provide a more detailed comparison of the convergence speed of the five algorithms, we present the average number of iterations the five methods required to reach a certain target in Tables 11 to 15. These targets were linearly selected between the worst and the best solutions found for each instance with 0% representing the worst and 100% the best solution.



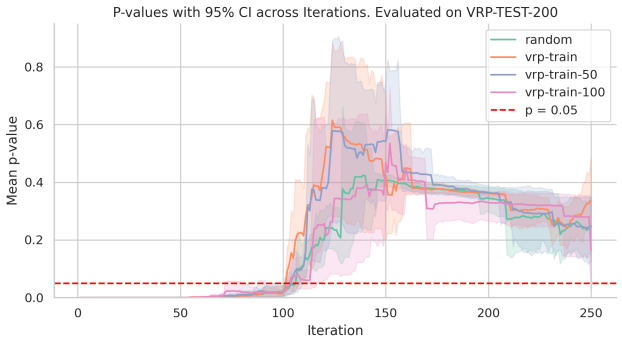
(a) VRP-TEST-20



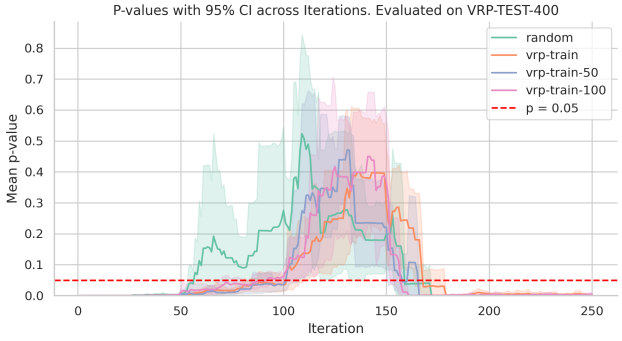
(b) VRP-TEST-50



(c) VRP-TEST-100



(d) VRP-TEST-200



(e) VRP-TEST-400

Figure 4: Pairwise p-values between each learning-based model combination and the standard HGS algorithm for different VRP test sizes, calculated using the two-sided T-test. Values below 0.05 indicate statistically significant differences between the two solutions.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	32365.40 (+65.7%)	21134.33 (+20.5%)	20833.37 (+19.5%)	21056.82 (+21.0%)	21425.29 (+22.1%)
2	25497.88 (+32.1%)	19382.64 (+11.3%)	19279.55 (+10.9%)	19332.00 (+11.5%)	19610.45 (+12.3%)
3	23074.55 (+22.4%)	18747.16 (+8.3%)	18685.28 (+8.2%)	18708.61 (+8.5%)	18876.41 (+8.8%)
4	21826.79 (+17.7%)	18415.40 (+6.8%)	18368.55 (+6.8%)	18347.79 (+6.8%)	18521.00 (+7.2%)
5	21128.10 (+14.9%)	18187.66 (+5.8%)	18163.49 (+5.9%)	18140.08 (+5.9%)	18300.76 (+6.3%)
10	19661.48 (+9.4%)	17770.02 (+4.0%)	17783.09 (+4.1%)	17751.33 (+4.0%)	17832.97 (+4.2%)
20	18789.85 (+6.1%)	17555.89 (+2.9%)	17570.49 (+3.1%)	17541.93 (+2.9%)	17610.05 (+3.1%)
50	18097.20 (+3.7%)	17360.68 (+1.9%)	17362.66 (+2.0%)	17342.12 (+1.9%)	17369.10 (+1.9%)
100	17550.98 (+1.9%)	17197.73 (+1.1%)	17194.29 (+1.1%)	17195.72 (+1.2%)	17210.35 (+1.1%)
150	17252.43 (+1.0%)	17133.33 (+0.8%)	17130.37 (+0.8%)	17135.00 (+0.8%)	17129.36 (+0.8%)
200	17140.22 (+0.6%)	17097.36 (+0.6%)	17096.40 (+0.6%)	17096.93 (+0.6%)	17097.07 (+0.6%)
250	17078.38 (+0.5%)	17074.20 (+0.5%)	17073.99 (+0.5%)	17074.22 (+0.5%)	17074.27 (+0.5%)
Avg	+14.66%	+5.37%	+5.28%	+5.47%	+5.74%

Table 4: Average solution cost across all instances over 1 run at selected iterations, for the HGS algorithm and Hybrid variants, including the average gap to the optimal solutions. Evaluated on the XML-100 benchmark set.

All Hybrid-GAs exhibited similar convergence behaviour. Our observations indicate that the gap in convergence speed between the baseline and hybrid algorithms depends on the problem dimensionality, with the HGS algorithm converging noticeably slower. Furthermore, as observed in Section 6.1.1, the MLGA variants consistently start with better solutions and improve them by a relatively small amount, compared to the baseline algorithm. Ultimately, all algorithms eventually reached nearly identical solutions, which were close to optimal in the case of the XML-100 dataset.

6.2 Training Causes Slight Improvement

To further investigate the differences among the four MLGA variants, we present Figure 5. Figure 5a provides the results acquired on the VRP-TEST-20 set. We note that the randomly initialized model performed the worst on average for the first $\sim$ 200 iterations. On the VRP-TEST-50 set, the random model converged slightly slower in the first 100 iterations, but performed almost identical to the other models later on. In Figure 5c, the VRP-TRAIN-20-model converged slightly slower for the first $\sim$ 50 iterations, and caught up to the other algorithms afterwards. Figure 3d shows that the randomly initialized and VRP-TRAIN-20 model performed slightly worse, and both converged to follow the general trend line set by the other models after $\sim$ 100 iterations. Finally, in Figure 5e the random model performed slightly worse for the first $\sim$ 75 iterations, after which all models converged to a similar average solution.

Comparing the Hybrid-MLGAs to each other, we conclude that the random-MLGA performed the worst by a slight amount. As seen in Tables 5 to 9, the difference in performance varies by at most 5.0%, compared to the best performer. This shows that training the model slightly improves the performance of the Hybrid-MLGA. All algorithms converged at the same rate, except for the random-model on the VRP-TEST-20 set, where it performed slightly worse than the other algorithms for the first $\sim$ 200 iterations.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	6925.56	5261.09(-24.0%)	5120.44(-26.1%)	5164.49(-25.4%)	5202.98(-24.9%)
2	6177.59	5070.41(-17.9%)	4981.42(-19.4%)	5014.69(-18.8%)	5052.72(-18.2%)
3	5945.01	5032.16(-15.4%)	4957.81(-16.6%)	4984.74(-16.2%)	4995.61(-16.0%)
4	5835.30	4992.72(-14.4%)	4945.07(-15.3%)	4961.76(-15.0%)	4964.82(-14.9%)
5	5786.66	4977.67(-14.0%)	4937.97(-14.7%)	4945.88(-14.5%)	4956.94(-14.3%)
10	5717.02	4961.87(-13.2%)	4911.95(-14.1%)	4930.96(-13.7%)	4931.67(-13.7%)
15	5713.73	4958.70(-13.2%)	4902.77(-14.2%)	4916.21(-14.0%)	4920.41(-13.9%)
25	5697.81	4955.19(-13.0%)	4893.75(-14.1%)	4905.35(-13.9%)	4898.59(-14.0%)
50	5658.91	4942.58(-12.7%)	4889.11(-13.6%)	4893.09(-13.5%)	4896.97(-13.5%)
100	5438.68	4919.31(-9.5%)	4875.32(-10.4%)	4884.27(-10.2%)	4883.03(-10.2%)
150	5237.72	4912.60(-6.2%)	4872.05(-7.0%)	4879.33(-6.8%)	4875.77(-6.9%)
200	4985.52	4891.88(-1.9%)	4871.99(-2.3%)	4878.48(-2.1%)	4875.18(-2.2%)
250	4855.14	4855.80(+0.0%)	4857.09(+0.0%)	4856.11(+0.0%)	4856.78(+0.0%)
Avg		-12.0%	-12.9%	-12.6%	-12.5%

Table 5: Average solution cost across all instances over 5 runs at selected iterations for the HGS algorithm and hybrid variants, including the difference relative to HGS. Evaluated on the VRP-TEST-20 set.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	18081.02	11864.80(-34.4%)	11414.90(-36.9%)	11012.24(-39.1%)	11173.06(-38.2%)
2	15038.85	11118.93(-26.1%)	11003.34(-26.8%)	10619.66(-29.4%)	10658.32(-29.1%)
3	13924.40	10809.64(-22.4%)	10729.59(-22.9%)	10480.20(-24.7%)	10446.84(-25.0%)
4	13398.28	10690.23(-20.2%)	10583.31(-21.0%)	10398.42(-22.4%)	10357.02(-22.7%)
5	13092.51	10595.63(-19.1%)	10514.77(-19.7%)	10340.18(-21.0%)	10314.95(-21.2%)
10	12202.17	10355.30(-15.1%)	10301.01(-15.6%)	10206.43(-16.4%)	10197.18(-16.4%)
15	11975.75	10302.66(-14.0%)	10201.09(-14.8%)	10153.56(-15.2%)	10160.49(-15.2%)
25	11712.77	10156.70(-13.3%)	10143.47(-13.4%)	10114.36(-13.6%)	10082.05(-13.9%)
50	11576.43	10081.77(-12.9%)	10046.66(-13.2%)	10036.61(-13.3%)	10027.33(-13.4%)
100	10616.39	9936.36(-6.4%)	9952.81(-6.3%)	9920.52(-6.6%)	9952.19(-6.3%)
150	9935.28	9858.88(-0.8%)	9862.92(-0.7%)	9874.55(-0.6%)	9866.49(-0.7%)
200	9847.95	9852.77(+0.0%)	9849.72(+0.0%)	9860.36(+0.1%)	9852.78(+0.0%)
250	9843.56	9846.81(+0.0%)	9845.17(+0.0%)	9848.60(+0.1%)	9847.86(+0.0%)
Avg		-14.2%	-14.7%	-15.5%	-15.5%

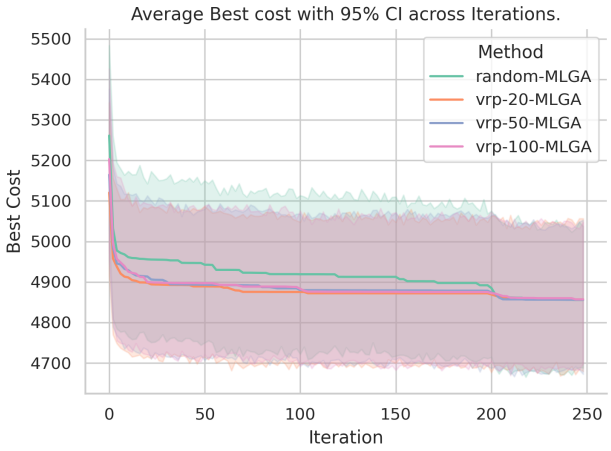
Table 6: Average solution cost across all instances over 5 runs at selected iterations for the HGS algorithm and hybrid variants, including the difference relative to HGS. Evaluated on the VRP-TEST-50 set.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	29605.09	18676.02(-36.9%)	18112.04(-38.8%)	18169.01(-38.6%)	17885.34(-39.6%)
2	23782.60	17248.19(-27.5%)	17242.91(-27.5%)	16966.39(-28.7%)	16702.44(-29.8%)
3	21389.53	16807.14(-21.4%)	16773.82(-21.6%)	16519.63(-22.8%)	16477.74(-23.0%)
4	20087.44	16468.43(-18.0%)	16505.82(-17.8%)	16236.12(-19.2%)	16359.35(-18.6%)
5	18869.97	16236.69(-14.0%)	16319.62(-13.5%)	16127.80(-14.5%)	16230.27(-14.0%)
10	17192.64	15991.14(-7.0%)	16032.51(-6.7%)	15919.37(-7.4%)	15944.59(-7.3%)
15	16695.90	15835.32(-5.2%)	15949.66(-4.5%)	15823.94(-5.2%)	15847.70(-5.1%)
25	16339.25	15709.14(-3.9%)	15803.48(-3.3%)	15752.04(-3.6%)	15743.55(-3.6%)
50	16186.83	15602.93(-3.6%)	15614.91(-3.5%)	15642.06(-3.4%)	15596.81(-3.6%)
100	15712.94	15447.66(-1.7%)	15483.79(-1.5%)	15462.23(-1.6%)	15489.24(-1.4%)
150	15406.05	15404.62(-0.0%)	15412.71(+0.0%)	15420.05(+0.1%)	15416.75(+0.1%)
200	15382.82	15382.29(-0.0%)	15388.84(+0.0%)	15391.81(+0.1%)	15389.66(+0.0%)
250	15364.59	15364.60(+0.0%)	15366.86(+0.0%)	15373.66(+0.1%)	15369.83(+0.0%)
Avg		-10.7%	-10.7%	-11.1%	-11.2%

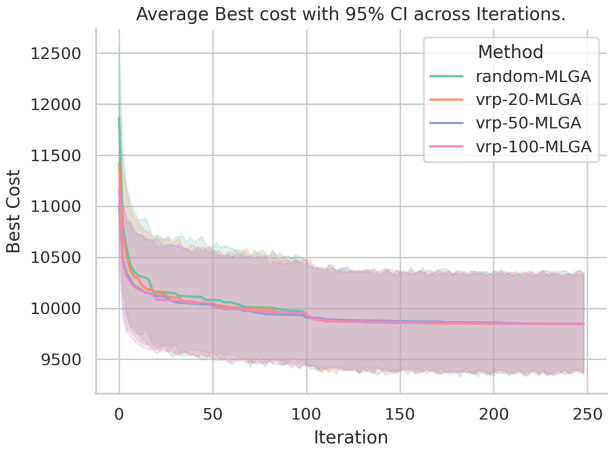
Table 7: Average solution cost across all instances over 5 runs at selected iterations for the HGS algorithm and hybrid variants, including the difference relative to HGS. Evaluated on the VRP-TEST-100 set.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	73507.04	44769.09(-39.1%)	42001.84(-42.9%)	41277.97(-43.8%)	41088.61(-44.1%)
2	55821.82	39642.08(-29.0%)	39278.54(-29.6%)	38356.53(-31.3%)	38109.82(-31.7%)
3	49743.35	38076.71(-23.5%)	37556.70(-24.5%)	36974.24(-25.7%)	37230.42(-25.2%)
4	46696.87	37282.56(-20.2%)	37204.99(-20.3%)	36262.95(-22.3%)	36644.42(-21.5%)
5	45224.17	36775.39(-18.7%)	36927.58(-18.3%)	36040.79(-20.3%)	36239.72(-19.9%)
10	40966.75	35980.74(-12.2%)	35826.41(-12.5%)	35493.43(-13.4%)	35657.18(-13.0%)
15	40091.87	35662.45(-11.0%)	35514.28(-11.4%)	35316.22(-11.9%)	35398.75(-11.7%)
25	39496.61	35423.80(-10.3%)	35326.64(-10.6%)	35104.21(-11.1%)	35181.34(-10.9%)
50	38242.84	34999.67(-8.5%)	35145.80(-8.1%)	34887.15(-8.8%)	34894.52(-8.8%)
100	36537.47	34825.97(-4.7%)	34732.53(-4.9%)	34514.89(-5.5%)	34597.23(-5.3%)
150	34304.33	34332.07(+0.1%)	34330.97(+0.1%)	34323.15(+0.1%)	34321.38(+0.0%)
200	34246.26	34264.72(+0.1%)	34267.62(+0.1%)	34257.77(+0.0%)	34258.61(+0.0%)
250	34200.97	34216.84(+0.0%)	34217.17(+0.0%)	34207.84(+0.0%)	34212.96(+0.0%)
Avg		-13.6%	-14.1%	-14.9%	-14.8%

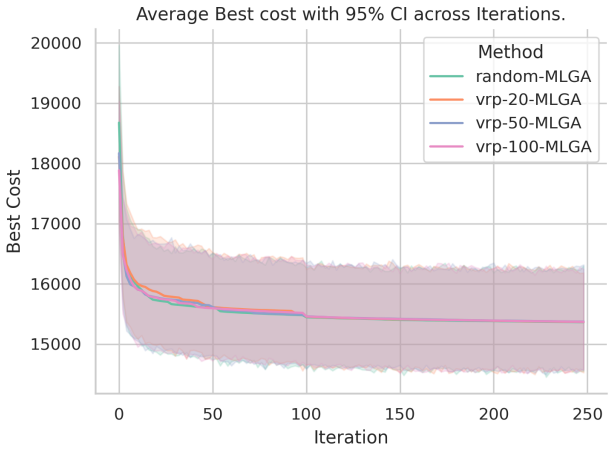
Table 8: Average solution cost across all instances over 5 runs at selected iterations for the HGS algorithm and hybrid variants, including the difference relative to HGS. Evaluated on the VRP-TEST-200 set.



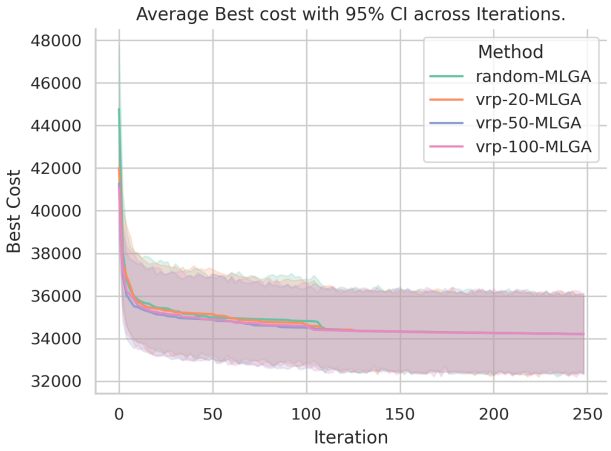
(a) VRP-TEST-20



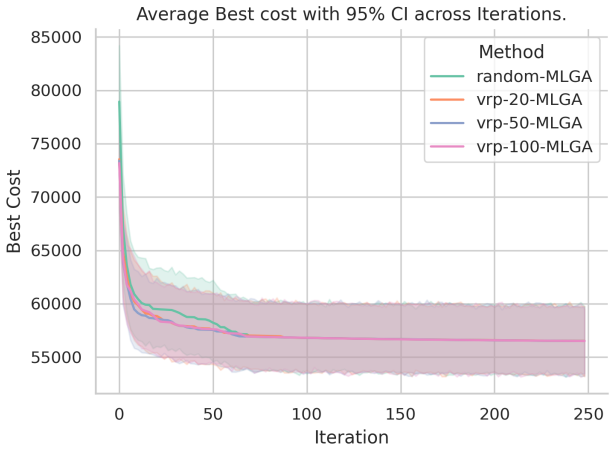
(b) VRP-TEST-50



(c) VRP-TEST-100



(d) VRP-TEST-200



(e) VRP-TEST-400

Figure 5: Average best cost per iteration, shown as the mean line with 95% CI for each problem size. Only Hybrid-MLGA variants are shown, where the prefix denotes the model used to initialize the population of the HGS.

Iter	HGS	random	vrp-train	vrp-train-50	vrp-train-100
1	115962.31	78932.61(-31.9%)	73535.25(-36.6%)	73397.36(-36.7%)	73169.03(-36.9%)
2	87171.50	70528.94(-19.1%)	66974.61(-23.2%)	66536.06(-23.7%)	67051.53(-23.1%)
3	77785.27	67141.26(-13.7%)	64812.52(-16.7%)	63897.56(-17.9%)	64169.37(-17.5%)
4	73902.61	65405.37(-11.5%)	63148.88(-14.6%)	62831.58(-15.0%)	62781.65(-15.0%)
5	72035.44	63628.25(-11.7%)	62599.25(-13.1%)	61934.52(-14.0%)	61805.84(-14.2%)
10	66373.50	60618.24(-8.7%)	60054.05(-9.5%)	59237.29(-10.8%)	60114.66(-9.4%)
15	64318.31	59912.08(-6.9%)	59095.80(-8.1%)	58880.43(-8.5%)	59325.57(-7.8%)
25	61738.12	59453.76(-3.7%)	58391.56(-5.4%)	58472.71(-5.3%)	58298.95(-5.6%)
50	58447.89	58228.94(-0.4%)	57568.50(-1.5%)	57550.36(-1.5%)	57611.19(-1.4%)
100	56716.89	56808.69(+0.2%)	56805.52(+0.2%)	56808.28(+0.2%)	56810.15(+0.2%)
150	56610.12	56681.41(+0.1%)	56677.20(+0.1%)	56680.03(+0.1%)	56668.35(+0.1%)
200	56528.70	56581.64(+0.1%)	56589.82(+0.1%)	56577.77(+0.1%)	56575.61(+0.1%)
250	56464.19	56510.37(+0.1%)	56515.57(+0.1%)	56507.77(+0.1%)	56507.14(+0.1%)
Avg		-8.2%	-9.9%	-10.2%	-10.0%

Table 9: Average solution cost across all instances over 5 runs at selected iterations for the HGS algorithm and hybrid variants, including the difference relative to HGS. Evaluated on the VRP-TEST-400 set.

6.3 Generalizes to New Dimensions

As discussed previously, the models trained on a certain size of instances initially perform the best on unseen instances of the same size. Notably, even if they did not perform the best at the start, all models generalize to unseen problem sizes, having a similar solution quality as the "specialized" models. To further investigate this, we review the results for the VRP-TEST-200 and VRP-TEST-400 sets. These datasets are of a size no model is trained on, thus providing a good indicator of the performance of the MLGAs on problems of unseen dimensions. We notice that the larger the dimension, the less spread out the solutions of the four variants of the model become. In Figure 5d, the best performer is the model trained on a dimension closest to that of the VRP-TEST-200 set, which is the VRP-TRAIN-100-model, and the same is seen for the results presented in Figure 5e. We conclude that the GCN-NPEC model generalizes well to unseen dimensions, and delivers robust performance, even when untrained.

6.4 Hybrid-MLGA Delivers Better Anytime Performance

While reporting the average performance over a whole test set provides a useful overall comparison between methods, it may not fully capture the differences across individual instances, which can vary in characteristics such as route length or customer distribution. To address this limitation, we additionally provide the Aggregated Empirical Cumulative Distribution Functions (ECDFs) for all five methods.

The ECDF shows the cumulative fraction of $(\text{run}, \text{function}, \text{target})$ triples that reach the target cost v as a function of the number of iterations t. At each t, the ECDF value corresponds to the proportion of runs for which the best-found solution is at least as good as the target. A higher

ECDF value indicates better performance, as a greater fraction of runs achieved the target value with fewer iterations. A steeper curve implies more stable behaviour, as most runs reach a target with the same iteration budget. A left-shifted curve shows a faster convergence rate, and a higher curve indicates a larger success rate.

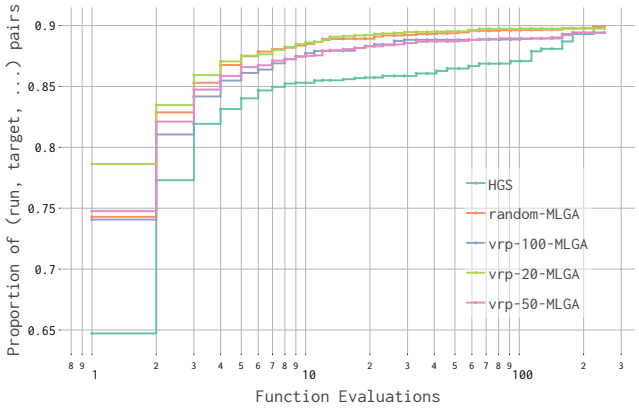
For this experiment, we generate 10 targets for each instance, linearly selected between the worst and best solutions reached for the relevant instance. We use the IOHanalyzer [DWY+18] web application to generate the aggregated ECDF plots, treating each instance as an individual function, and present these curves in Figure 6. Note that instances where the targets were identical are excluded from the aggregation, as they would unfairly skew the results. In such cases, the ECDF values would always be 1.0; therefore, the maximum proportion in Figure 6 does not reach 1.0 at 250 iterations, as would be expected if those instances were included.

For the VRP-TEST-20, VRP-TEST-50, VRP-TEST-100, and VRP-TEST-200 sets, the ECDF curves for the HGS algorithm are less steep compared to those of the other algorithms, suggesting less stable behaviour and greater variance in performance. Notable are the results in Figure 6c, which shows that the HGS is less stable for only the first ~ 15 iterations, after which all algorithms are identically stable, and the results in Figure 6e, which show that the HGS is more stable than the hybrid algorithms after the first ~ 10 iterations. Comparing the stability of the MLGAs to each other, Figure 6a shows that the VRP-TRAIN-20-model is slightly more stable and has a faster convergence rate compared to the other algorithms, and Figure 6b shows that the VRP-TRAIN-50-model has the fastest convergence rate and highest stability in the first 10 iterations, and is overtaken by the VRP-TRAIN-20 model afterwards. In Figure 6c, the four algorithms have the almost exact same convergence rate and stability, and Figure 6d shows that the VRP-TRAIN-50-model performs better than the VRP-TRAIN-100-model in both convergence rate and stability. Finally, Figure 6e shows that the models follow an almost identical ECDF curve. From these observations we can conclude that the VRP-TRAIN-50-model is slightly more stable and converges slightly faster than the other models, and that the Hybrid-GAs are more stable and converge faster than the baseline algorithm, even on unseen and untrained problem sizes. The Hybrid-MLGAs thus provide better anytime performance.

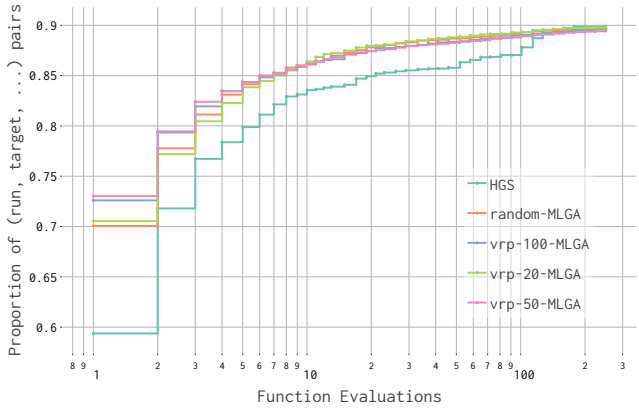
6.5 Hybrid-MLGA Reaches Targets Faster

In the previous sections, we have shown that the HGS algorithm converges slower than the hybrid algorithms. To expand on this, we provide the average runtimes of the five algorithms and the average inference times of the four models for each test set in Table 10. We note that the random-model, VRP-TRAIN-20-model, and VRP-TRAIN-100-model have similar run times per iteration, and that the VRP-TRAIN-50-model takes slightly longer compared to the other models, on all datasets. The runtimes of the HGS are slightly longer compared to the MLGAs and the inference times start much lower than the runtimes but continue to increase the larger the dimensionality of the problem set becomes. For the VRP-TEST-200 set, the inference time is still inside the standard deviation of the runtime, except for the VRP-TRAIN-100 model. For 400 node instances, the inference time increased to be greater than the algorithm runtime.

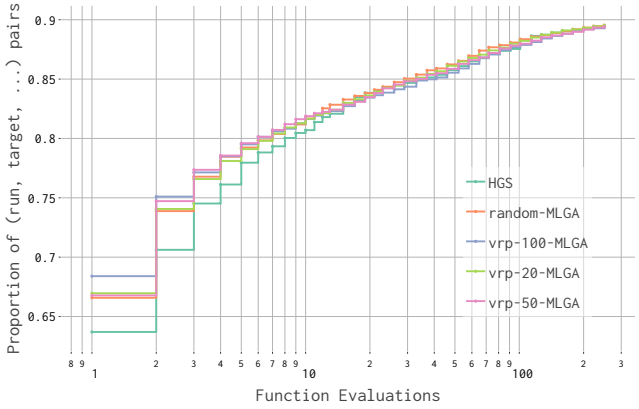
In addition to the generated test sets, we compare the performance of the baseline algorithm with the Hybrid-MLGAs on the XML-100 benchmark set in Table 4. These results, consistent with earlier findings, further demonstrate that the Hybrid-GAs converged faster compared to the baseline algorithm.



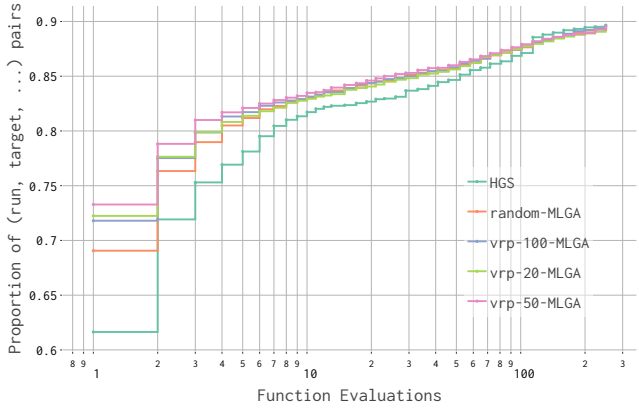
(a) VRP-TEST-20



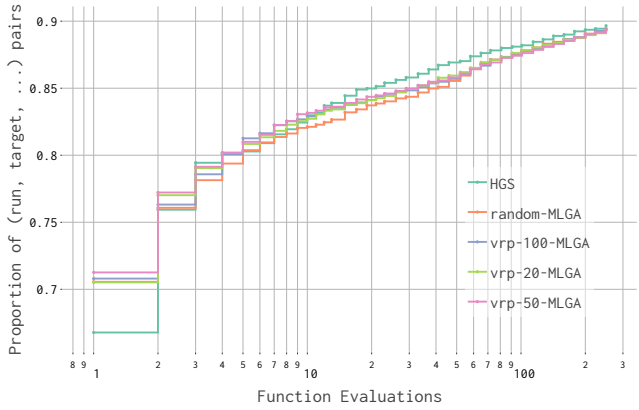
(b) VRP-TEST-50



(c) VRP-TEST-100



(d) VRP-TEST-200



(e) VRP-TEST-400

Figure 6: Aggregated Empirical Cumulative Distribution Functions (ECDFs) over all tested problem instances and runs in the respective test sets. Each plot shows the fraction of (run, instance, target) triples for which the best solution found within a given evaluation budget achieves at least a certain quality level v. Results are aggregated over all instances and all 5 runs per instance. Generated using IOHanalyzer [DWY+18].

Dim	Type	HGS	random	train	train-50	train-100
n20	RT	0.033 ± 0.010	0.0290 ± 0.0054	0.0295 ± 0.0057	0.0466 ± 0.0083	0.0261 ± 0.0060
	INF	–	0.00063 ± 0.00002	0.00063 ± 0.00002	0.00099 ± 0.00004	0.00060 ± 0.00002
n50	RT	0.038 ± 0.007	0.0355 ± 0.0053	0.0346 ± 0.0049	0.0465 ± 0.0067	0.0343 ± 0.0058
	INF	–	0.00329 ± 0.00009	0.00328 ± 0.00007	0.00505 ± 0.00016	0.00325 ± 0.00007
n100	RT	0.044 ± 0.008	0.0400 ± 0.0061	0.0410 ± 0.0060	0.0511 ± 0.0080	0.0407 ± 0.0061
	INF	–	0.0137 ± 0.0004	0.0139 ± 0.0001	0.0211 ± 0.0005	0.0170 ± 0.0016
n200	RT	0.056 ± 0.020	0.0470 ± 0.0112	0.0498 ± 0.0119	0.0759 ± 0.0243	0.0499 ± 0.0117
	INF	–	0.0527 ± 0.0007	0.0518 ± 0.0013	0.0660 ± 0.0046	0.0635 ± 0.0058
n400	RT	0.163 ± 0.071	0.1376 ± 0.0521	0.1420 ± 0.0531	0.1903 ± 0.0642	0.1766 ± 0.0711
	INF	–	0.2018 ± 0.0082	0.1969 ± 0.0077	0.2549 ± 0.0687	0.2584 ± 0.0342

Table 10: Average runtime (RT) per iteration (in seconds) for HGS and Combi, and inference time (INF) per instance across different model variants: `random`, `vrp-train`, `vrp-train-50`, and `vrp-train-100`. Each value is given with its standard deviation.

When considered alongside the runtimes and inference times presented in Table 10, and the difference in iterations needed to reach a certain target shown in Tables 11 to 15, we conclude that the best performing Hybrid-MLGA achieves a lower overall runtime to reach a certain target compared to the standard HGS algorithm, across all problem sizes. Note that this includes the overhead of generating the 25 solutions used to initialize the HGS population.

% to best cost	HGS	random	vrp-train	vrp-train-50	vrp-train-100
0%	0.00	0.00	0.00	0.00	0.00
10%	3.80	0.12 (-3.68)	0.08 (-3.72)	0.09 (-3.71)	0.09 (-3.72)
20%	3.81	0.17 (-3.64)	0.14 (-3.68)	0.13 (-3.68)	0.17 (-3.64)
30%	4.47	0.25 (-4.22)	0.15 (-4.32)	0.18 (-4.29)	0.24 (-4.23)
40%	4.51	0.32 (-4.19)	0.16 (-4.35)	0.60 (-3.91)	0.34 (-4.17)
50%	5.11	0.38 (-4.73)	0.18 (-4.93)	0.67 (-4.44)	0.48 (-4.63)
60%	5.68	0.54 (-5.14)	0.30 (-5.38)	1.95 (-3.73)	2.10 (-3.58)
70%	5.90	1.16 (-4.74)	0.58 (-5.32)	2.52 (-3.38)	2.31 (-3.59)
80%	7.42	1.98 (-5.44)	1.04 (-6.38)	3.23 (-4.19)	2.57 (-4.85)
90%	8.43	5.21 (-3.23)	1.84 (-6.59)	4.08 (-4.36)	4.30 (-4.13)
100%	21.66	15.23 (-6.43)	12.58 (-9.08)	18.05 (-3.61)	16.78 (-4.88)

Table 11: Average number of iterations required to reach selected target solutions, evaluated across all instances over 5 runs for the HGS algorithm and hybrid variants, including the difference relative to HGS. Each target is linearly spaced apart, between the worst and best cost. Evaluated on the VRP-TEST-20 set.

% to best cost	HGS	random	vrp-train	vrp-train-50	vrp-train-100
0%	0.00	0.00	0.00	0.00	0.00
10%	2.59	0.06 (-2.53)	0.06 (-2.53)	0.02 (-2.56)	0.06 (-2.53)
20%	2.60	0.11 (-2.49)	0.09 (-2.52)	0.07 (-2.54)	0.08 (-2.52)
30%	3.52	0.19 (-3.33)	0.19 (-3.32)	0.12 (-3.40)	0.15 (-3.37)
40%	3.60	0.26 (-3.34)	0.58 (-3.03)	0.21 (-3.40)	0.22 (-3.39)
50%	4.64	0.40 (-4.24)	0.90 (-3.74)	0.47 (-4.17)	0.73 (-3.91)
60%	5.06	1.09 (-3.97)	1.17 (-3.89)	1.01 (-4.05)	0.96 (-4.10)
70%	5.56	1.77 (-3.79)	2.07 (-3.49)	1.77 (-3.79)	2.16 (-3.40)
80%	7.62	5.47 (-2.15)	4.81 (-2.81)	4.41 (-3.21)	5.34 (-2.28)
90%	13.15	13.89 (+0.75)	12.24 (-0.90)	11.73 (-1.42)	10.77 (-2.38)
100%	108.13	76.76 (-31.37)	70.07 (-38.05)	78.05 (-30.08)	76.51 (-31.62)

Table 12: Average number of iterations required to reach selected target solutions, evaluated across all instances over 5 runs for the HGS algorithm and hybrid variants, including the difference relative to HGS. Each target is linearly spaced apart, between the worst and best cost. Evaluated on the VRP-TEST-50 set.

% to best cost	HGS	random	vrp-train	vrp-train-50	vrp-train-100
0%	0.00	0.00	0.00	0.00	0.00
10%	0.66	0.09 (-0.57)	0.09 (-0.57)	0.04 (-0.62)	0.06 (-0.60)
20%	0.67	0.13 (-0.54)	0.13 (-0.53)	0.07 (-0.60)	0.10 (-0.57)
30%	0.91	0.19 (-0.73)	0.23 (-0.68)	0.14 (-0.77)	0.16 (-0.76)
40%	1.04	0.35 (-0.70)	0.38 (-0.67)	0.34 (-0.71)	0.28 (-0.76)
50%	1.71	0.73 (-0.98)	0.76 (-0.95)	0.60 (-1.11)	0.68 (-1.03)
60%	2.22	1.49 (-0.74)	1.39 (-0.83)	1.37 (-0.86)	1.62 (-0.60)
70%	4.05	3.94 (-0.11)	3.59 (-0.46)	4.00 (-0.05)	4.22 (+0.17)
80%	9.50	10.41 (+0.91)	10.55 (+1.05)	11.07 (+1.57)	11.04 (+1.54)
90%	26.62	24.21 (-2.41)	25.70 (-0.92)	26.87 (+0.25)	26.52 (-0.10)
100%	186.00	158.40 (-27.60)	153.02 (-32.98)	153.04 (-32.96)	182.23 (-3.77)

Table 13: Average number of iterations required to reach selected target solutions, evaluated across all instances over 5 runs for the HGS algorithm and hybrid variants, including the difference relative to HGS. Each target is linearly spaced apart, between the worst and best cost. Evaluated on the VRP-TEST-100 set.

% to best cost	HGS	random	vrp-train	vrp-train-50	vrp-train-100
0%	0.00	0.00	0.00	0.00	0.00
10%	1.45	0.09 (-1.36)	0.04 (-1.41)	0.01 (-1.44)	0.04 (-1.41)
20%	1.46	0.16 (-1.31)	0.07 (-1.39)	0.03 (-1.43)	0.08 (-1.38)
30%	1.97	0.26 (-1.71)	0.18 (-1.78)	0.13 (-1.84)	0.19 (-1.78)
40%	2.08	0.39 (-1.69)	0.34 (-1.74)	0.23 (-1.85)	0.32 (-1.76)
50%	2.70	0.87 (-1.83)	0.75 (-1.95)	0.61 (-2.10)	0.63 (-2.07)
60%	3.36	2.21 (-1.15)	1.70 (-1.66)	1.57 (-1.79)	1.74 (-1.62)
70%	5.42	5.84 (+0.41)	5.05 (-0.37)	4.92 (-0.51)	5.08 (-0.34)
80%	13.78	15.14 (+1.36)	14.76 (+0.98)	13.92 (+0.14)	14.79 (+1.01)
90%	29.05	24.10 (-4.95)	23.56 (-5.49)	25.41 (-3.64)	25.95 (-3.10)
100%	236.83	213.59 (-23.24)	196.65 (-40.19)	162.85 (-73.98)	192.63 (-44.20)

Table 14: Average number of iterations required to reach selected target solutions, evaluated across all instances over 5 runs for the HGS algorithm and hybrid variants, including the difference relative to HGS. Each target is linearly spaced apart, between the worst and best cost. Evaluated on the VRP-TEST-200 set.

% to best cost	HGS	random	vrp-train	vrp-train-50	vrp-train-100
0%	0.00	0.00	0.00	0.00	0.00
10%	0.49	0.14 (-0.34)	0.10 (-0.39)	0.06 (-0.43)	0.07 (-0.42)
20%	0.50	0.21 (-0.28)	0.13 (-0.37)	0.11 (-0.39)	0.11 (-0.38)
30%	0.77	0.40 (-0.38)	0.25 (-0.53)	0.18 (-0.60)	0.24 (-0.54)
40%	0.84	0.70 (-0.14)	0.57 (-0.27)	0.41 (-0.43)	0.47 (-0.37)
50%	1.47	1.34 (-0.13)	1.21 (-0.26)	1.02 (-0.45)	1.28 (-0.19)
60%	2.01	3.78 (+1.77)	3.08 (+1.07)	2.71 (+0.70)	2.74 (+0.73)
70%	4.43	7.53 (+3.10)	7.33 (+2.90)	6.79 (+2.36)	6.47 (+2.04)
80%	11.82	16.39 (+4.57)	16.59 (+4.77)	17.13 (+5.31)	15.91 (+4.09)
90%	27.73	27.35 (-0.38)	23.96 (-3.78)	26.50 (-1.24)	25.88 (-1.85)
100%	237.60	214.46 (-23.14)	208.83 (-28.77)	225.75 (-11.85)	193.55 (-44.05)

Table 15: Average number of iterations required to reach selected target solutions, evaluated across all instances over 5 runs for the HGS algorithm and hybrid variants, including the difference relative to HGS. Each target is linearly spaced apart, between the worst and best cost. Evaluated on the VRP-TEST-400 set.

6.6 Summary

In the previous sections, we discussed the difference between the HGS algorithm and our Hybrid-MLGA. We concluded that the HGS performed worse than the hybrid method, up to a certain iteration budget, and above this budget all algorithms eventually reach a similar average solution quality for all tested dimensions of problems. The final solutions reached are close to optimal, with only a 0.5% gap. The models trained on a certain size of instances generated better initial solutions for instances of the same size. Moreover, the closer the dimensionality of the training instances to the problem size, the better the generated solutions, even on unseen dimensions. The GCN-NPEC model generalized well to other dimensions. The untrained model performed very close to the trained models and much better than the standard HGS algorithm. The Hybrid-MLGAs are more stable and converged faster or similar compared to the baseline algorithm, with the exception of the n400 problem set, where the HGS was slightly more stable at iterations greater than 15. Overall, the Hybrid-MLGA provided better anytime performance compared to the HGS algorithm, and the hybrid-GA reached a similar solution quality faster. Finally, we have shown that there exists a statistically significant performance gap between randomly initializing the HGS algorithm and providing initial machine learned solutions to fill the population of the HGS, which allowed the Hybrid-MLGA to converge faster.

7 Future Work

Due to time and computational constraints, we evaluated the performance gap using only one model and one Genetic Algorithm. As a result, it remains uncertain whether our results generalize to other models or algorithms. Future research could investigate this relation by evaluating multiple state-of-the-art meta-heuristic algorithms to assess if a performance gap persists. Additionally, comparing multiple solution generating methods could provide insights into how different methods influence the final algorithm performance.

Applying hyper parameter optimization (HYPO) to either the HGS and/or GCN-NPEC model may also influence the performance of the Hybrid-MLGA. Finally, specifically for the GCN-NPEC model, generating solutions using the edge probability matrix provided by the classification decoder may provide better solutions than the solutions produced by the sequential decoder, or training the model for a larger amount of epochs may further improve performance.

References

[AB17] Juho Andelmin and Enrico Bartolini. An exact algorithm for the green vehicle routing problem. *Transportation Science*, 51(4):1288–1303, 2017.

[AYH+24] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias

Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024.

[BA03] Barrie M. Baker and M.A. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[BBCV21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

[BCC+21] Ruibin Bai, Xinan Chen, Zhi-Long Chen, Tianxiang Cui, Shuhui Gong, Wentao He, Xiaoping Jiang, Huan Jin, Jiahuan Jin, Graham Kendall, Jiawei Li, Zheng Lu, Jianfeng Ren, Paul Weng, Ning Xue, and Huayan Zhang. Analytics and machine learning in vehicle routing research, 2021.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[BLS13] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013. Prediction, Control and Diagnosis using Advanced Neural Computations.

[BM04] John E. Bell and Patrick R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48, 2004.

[BMMD24] Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 25(6):4754–4772, 2024.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[CW64] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[DBS06] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[DG97a] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DG97b] Marco Dorigo and Luca Maria Gambardella. Ant colonies for the travelling salesman problem. *Biosystems*, 43(2):73–81, 1997.

[DG99] Marco Dorigo and Luca Maria Gambardella. Ant algorithms for discrete optimization." artificial life 5, 137-172. *Artificial Life*, 5:137–172, 04 1999.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[DR59] George Dantzig and John Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[DWY+18] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. IOHprofiler: A Benchmarking and Profiling Tool for Iterative Optimization Heuristics. *arXiv e-prints:1810.05281*, October 2018.

[DZH+20] Lu Duan, Yang Zhan, Haoyuan Hu, Yu Gong, Jiangwen Wei, Xiaodong Zhang, and Yinghui Xu. Efficiently solving the practical vehicle routing problem: A novel joint learning approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3054–3063, New York, NY, USA, 2020. Association for Computing Machinery.

[EA20] 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.

[FL19] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, May 2019.

[Hel17] Keld Helsgaun. *An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems: Technical report.* Roskilde Universitet, December 2017.

[Hol73] John H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[HYL17] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[JST08] Nicolas Jozefowiez, Frédéric Semet, and El-Ghazali Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309, 2008.

[KCK21] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80:8091–8126, 2021.

[KvHW19] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[LLYL09] Shih-Wei Lin, Zne-Jung Lee, Kuo-Ching Ying, and Chou-Yuan Lee. Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*, 36(2, Part 1):1505–1512, 2009.

[LLZ25] Xu Li, Zhengyan Liu, and Yan Zhang. Research on vehicle routing problem with time window based on improved genetic algorithm. *Scalable Computing: Practice and Experience*, 26(1):123–135, 2025.

[LOL09] Maly LOLek. Vehicle routing problem example. https://commons.wikimedia.org/wiki/File:Vehicle_Routing_Problem_Example.svg, 2009. Accessed: 2025-05-20.

[NDP+24] Peter Nielsen, Mahekha Dahanayaka, H.Niles Perera, Amila Thibbotuwawa, and Deniz Kenan Kilic. A systematic review of vehicle routing problems and models in multi-echelon distribution networks. *Supply Chain Analytics*, 7:100072, 2024.

[NOTS18] Mohammadreza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V. Snyder. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

[ORH06] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-Objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, February 2006.

[QSUV22] Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and Thibaut Vidal. 10,000 optimal CVRP solutions for testing machine learning based heuristics. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*, 2022.

[QZW+24] Dingding Qi, Yingjun Zhao, Zhengjun Wang, Wei Wang, Li Pi, and Longyue Li. Joint approach for vehicle routing problems based on genetic algorithm and graph convolutional network. *Mathematics*, 12(19), 2024.

[RMLG07] A. E. Rizzoli, R. Montemanni, E. Lucibello, and L. M. Gambardella. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1(2):135–151, 2007.

[SB98] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054, 1998.

[SGT+09] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[SPGM24] Reza Shahbazian, Luigi Di Puglia Pugliese, Francesca Guerriero, and Giusy Macrina. Integrating machine learning into vehicle routing problem: Methods and applications. *IEEE Access*, 12:93087–93115, 2024.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[TV02] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.

[TY21] Shi-Yi Tan and Wei-Chang Yeh. The vehicle routing problem: State-of-the-art classification and review. *Applied Sciences*, 11(21), 2021.

[VCGP13] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[VFJ17] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.

[Vid22] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[WCLY16] Xinyu Wang, Tsan-Ming Choi, Haikuo Liu, and Xiaohang Yue. Novel ant colony optimization methods for simplifying solution construction in vehicle routing problems. *IEEE Transactions on Intelligent Transportation Systems*, 17(11):3132–3141, 2016.

[WLK24] Niels A. Wouda, Leon Lan, and Wouter Kool. PyVRP: a high-performance VRP solver package. *INFORMS Journal on Computing*, 36(4):943–955, 2024.

[WWH+24] Rixin Wu, Ran Wang, Jie Hao, Qiang Wu, Ping Wang, and Dusit Niyato. Multiobjective vehicle routing optimization with time windows: A hybrid approach using deep reinforcement learning and nsga-ii, 2024.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[XLT+18] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.

[ZGYT22] Haifei Zhang, Hongwei Ge, Jinlong Yang, and Yubing Tong. Review of vehicle routing problems: Models, classification and solving algorithms. *Archives of Computational Methods in Engineering*, pages 1–27, 2022.

A Technical Specification

The following sections provide implementation details related to our adaption of the GCN-NPEC model.

A.1 Data loader

To represent a VRP instance as a graph, we utilized the PyTorch Geometric (PyG) library [FL19] in combination with the VRPLIB package [Hel17]. VRPLIB provides functionality for reading and writing VRP instances using a (semi-)standardized data format. Upon loading a VRP instance, it automatically calculates the edge lengths based on a specified method, such as Euclidean distance or provided distance matrices.

We implemented a custom `Dataloader`[3] class that parses VRPLIB instances into PyG `Data`[4] objects. Each `Data` object key components of a VRP graph, such as the demand and positions of the nodes, the edge lengths (also called edge weights), and the edge indices (which nodes are connected to which other nodes). Additionally, we pre-calculate the adjacency matrix A, as defined in Equation (3). A `Data` object represents a complete VRP instance, containing all the necessary information on its structure and parameters (fleet size, fleet capacity, etc.).

A.2 Encoding

As illustrated in Section 4.2.1, the graph encoder comprises of two main parts: the initial graph instance encoding and the subsequent GCN layers that extract the encoded features. In our implementation, we divide these parts into two classes: the `VRPInputEncoder` class and the `GCNLayer` class. The `VRPInputEncoder` closely follows the specification outlined in Section 4.2.1, although with certain deviations. Specifically, because the concatenation operation doubles the feature dimensions d_x and d_y, we take the output dimension of the encoded node and edge features to be $2 \cdot d_x$ and $2 \cdot d_y$, respectively. The resulting embeddings generated by Equation (5) and Equation (6) have a final dimension of d_h.

A key function of GCN layers is to update the graph node features via message passing within their neighbourhood. The GCN-NPEC model extends this concept by incorporating edge feature updates and enabling joint updates of the node and edge embeddings simultaneously. We implement these message passing networks with the PyG `MessagePassing` base class, which provides a message passing framework and is built to work with PyG `Data` objects. We calculate the aggregated attention scores for a node's neighbourhood $N(i)$ with the `MessagePassing.update` method and built-in aggregation method[5]. Similarly, the aggregated edge features over the edge neighbourhood $N(e_{i,j})$ are computed according to Equation (12).

The aggregated features are then combined in the forward pass of each GCN layer, and passed to the next layer of the GCN stack. Because both the node and edge embeddings are combined via the concatenation operation, it is essential to ensure each layer has consistent input and output

[3] https://pytorch-geometric.readthedocs.io/en/latest/modules/loader.html

[4] https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.data.Data.html

[5] https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.MessagePassing.html

dimensions, to support multi-layer GCN architectures. As briefly mentioned by the authors in [DZH+20], we add a skip-connection and layer normalization to each sublayer.

Finally, we integrate the `VRPInputEncoder` with L GCN layers in a single PyTorch `Module`. This module receives a VRP instance and outputs learned node and edge embeddings.

A.3 Decoding

We implement the L_{gru}-layered recurrent neural network (RNN) with the gated recurrent units (GRU), as described in Section 4.2.4, with PyTorch's GRU[6] module. We compute the last hidden state of the GRU module z_t as defined in Equation (18). The sequential decoder finally returns returns a vector representing the pointing probability distribution towards the nodes.

The classification decoder is relatively straightforward to implement. We stack L_{MLP} PyTorch Linear layers[7] to generate a L_{MLP}-layered Multilayer Perceptron (MLP). Although the authors do not specifically mention this, we apply a non-linear activation function (in our case the `ReLU` function) after each layer, excluding the final output layer. The classification decoder finally returns an edge probability matrix, as defined in Equation (20).

A.4 Generating a solution

To construct a viable solution $\boldsymbol{\pi}$, we follow the decoding procedure outlined in Algorithm 1 from the original work [DZH+20]. At each decoding step t, a customer node is selected to visit, while all infeasible nodes are masked. After visiting the node, the vehicle capacity, remaining demand and node mask N'_t are updated, as described in Section 4.2.4 The complete decoding loop is presented as pseudocode in Algorithm 2,.

[6] https://docs.pytorch.org/docs/stable/generated/torch.nn.GRU.html
[7] https://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html