Utrecht University

Faculty of Science

Thesis MSc Artificial Intelligence

Graph Neural Networks in Neighbourhood Selection for a Vehicle Routing Problem solver

Internal First Supervisors:
Dr. Mihaela Mitici
Algorithmic Data Analysis

Author:
Elitsa Bahovska
0071668

Internal Second Supervisors:
Dr. Thijs van Ommen
Algorithmic Data Analysis

External Supervisor:
Koen Dekker
Data Science Specialist

May 17, 2023

Abstract

This research paper examines the use of Graph Neural Networks (GNNs) for selecting a sub-problem to be optimized by a Vehicle Routing Problem (VRP) solver. The task is termed neighbourhood selection and has many overlapping properties with a more extensively researched approach to solving VRPs - Large Neighbourhood Search (LNS). This paper explores how do GNNs compare to other methods of neighbourhood selection for optimizing the solution of VRP which are already developed within Dassault Systèmes ' DELMIA, in collaboration with whom the current research is conducted.

VRP assumes a set of destinations that need to be visited by a set of vehicles and can be defined as a problem from graph theory. Because of this, GNN is considered a likely tool to approach solving the problem. In order to explore the method, sub-problems of VRP are represented as graphs, embedded via a GNN and compared in terms of how much optimizing them would improve the overall solution. Multiple models assuming different graph representations are trained on that task. They are consequently tested in real-world-like scenarios, where a VRPs solution is iteratively improved by using the models to select the best neighborhood (or sub-problem) which is then optimized using aDassault Systèmes ' algorithm.

Through experimentation, this research project examines methods to represent VRP sub-problems as a graphs-input for GNNs, as well as how to make and compare GNN embeddings. It provides a proof of concept that a GNN is a feasible approach to neighbourhood selection for VRP optimization.

Contents

Chapter 1

Introduction

This research paper examines the use of Graph Neural Networks (GNNs) in order to introduce a novel way of sub-problem selection for a Vehicle Routing Problem (VRP). Termed neighbourhood selection, it has many overlapping properties with a more extensively researched approach to solving VRPs, known as Large Neighbourhood Search (LNS). The core difference between the two is that, while LNS builds multiple solutions that only differ in a small section of the problem, neighbourhood selection choses a limited sub-problem focused solely on the changes to be optimized for a solution. Neighbourhood selection is usable in most of the conditions where LNS is, and has numerous applications: from real world usage (like logistics planning, such as solving VRP) to abstract points in n-dimensional space with various similarity measures [1], [29].

VRP deals with assigning multiple vehicles to visit points of interest in sequences, forming routes. This characteristic allows a VRP to be represented as a graph (described in Section 1.3), making it a suitable candidate for incorporation of GNNs in its solver. VRPs are further expanded upon in Section 1.1.

The benefit of discussing neighbourhood selection and its context of LNS within a VRP is twofold. First, it helps to constrain and understand the abstract concept of neighbourhood selection. This simplifies the explanation for the use and benefits of neighbourhood selection in a VRP its relation to LNS. With an established use in the solution of a VRP, it is easy to generalize back to other Combinatorial Optimisation Problems using LNS, primarily those in Logistics (VRP, Travelling Salesman, Vehicle Hailing Problems (VHP), etc.). Second, it allows for exploration of neighbourhood selection with specific graph representations of VRP subproblems.

The goal of this paper is to show that GNN proves invaluable in cleverly selecting neighborhoods as sub-problems to work on in order to solve a VRP.

This research is conducted in collaboration with Dassault Systèmes DELMIA, who have developed a tool to solve VRPs in various real world applications - the Logistics Planner (LP). LP optimizes solutions to transport planning problems grounded in VRP from the first to the last mile: such as the delivery of packages, home delivery of groceries, or restocking of warehouses or shops. This paper focuses on the delivery of goods to the final client in a real world context with DELMIA's product "Last Mile", an implementation of LP for last mile deliveries.

This chapter describe the VRP in detail, along with LP's way of solving it and the role of neighbourhood selection. The core problem is detailed: what needs to be done to select the best sub-problem for LP's sub-solver, using a GNN. Additionally, a literature review on the current research on LNS is provided, with common approaches to it, and what GNN add to the solution.

The subsequent chapters will discuss the GNN based approach that this research develops, its preliminaries, implementation and subsequent result.

1.1 Vehicle Routing Problem

A VRP considers delivering a set of shipments from an origin to a set of destinations. It is a Combinatorial Optimisation Problem and can be defined as finding an optimal set of routes for a fleet of delivery vehicles to traverse, from a depot to a set of customers. A **shipment** is generally defined by its delivery location and properties that the vehicle needs to accommodate. Potential extensions include adding other pickup locations, time windows, etc.

A **route** is a sequence of shipments (or the customers they need to be delivered to), that starts and ends at the depot. Within this research, it will be assumed that a route a consistent vehicle properties within its duration - e.g., the same capacity, a vehicle made to accommodate certain goods, etc. The route can pass through the depot multiple times to restock and deliver a new batch of shipments. A **trip** is a section of the route that starts and ends at a depot and does not visit it in-between.

A **solution** of a VRP is a set of routes for each vehicle, such that each shipment is assigned to a route. The goal of solving VRPs is to minimize their total cost.

1.1.1 VRP as a graph-theoretic problem

A VRP can also be represented as a graph theory problem [11]: Let $G = (V, E)$ be a graph, where the vertices V represent the locations a vehicle needs to visit. Vertex 0 corresponds to the depot and vertices $i = 1, ..., n$ - to the customers with a known demand. A non-negative cost c_{ij} is associated with each edge between two vertices, representing the travel cost from vertex i to vertex j.

The VRP consists of finding a collection of k cycles, each corresponding to a vehicle route (assuming one trip per route). Each circuit has a cost c_k defined as the sum of the costs of each edge. The following conditions need to be fulfilled.

1. Each circuit visits vertex 0 - the depot.
2. Each vertex $j \in V \setminus 0$ is visited by exactly one circuit.

Any such collection k is a solution. An **optimal solution** is one where $\sum_k c_k$ is minimal.

1.1.2 VRP with Capacity and Time Windows

This paper considers a specific variation of VRP with some extra constraints: VRP with *capacity* and *time windows* (also abbreviated VRPTW, or CVRPTW).

Capacity is a vehicle's transport capacity. In the graph theory based definition of the problem, this is represented with an additional condition:

3. Let each customer have a non-negative demand d_j. Then the sum of the demands of vertices on the same circuit must not exceed the vehicle capacity C. Formally: $\forall cir_k : \sum_j d_j < C_k$ for $j \in cir_k$, where cir_k is a member of the collection of sample circuits forming the solution k.

Time windows, added to the previously defined VRP, are additional shipment attributes, defining the two time points between which the delivery must happen. This is typically a hard time constraint [16]: a vehicle can arrive at a destination prior to the time window, but then waits for the start of the time window to deliver, which increases the virtual cost of the overall problem.

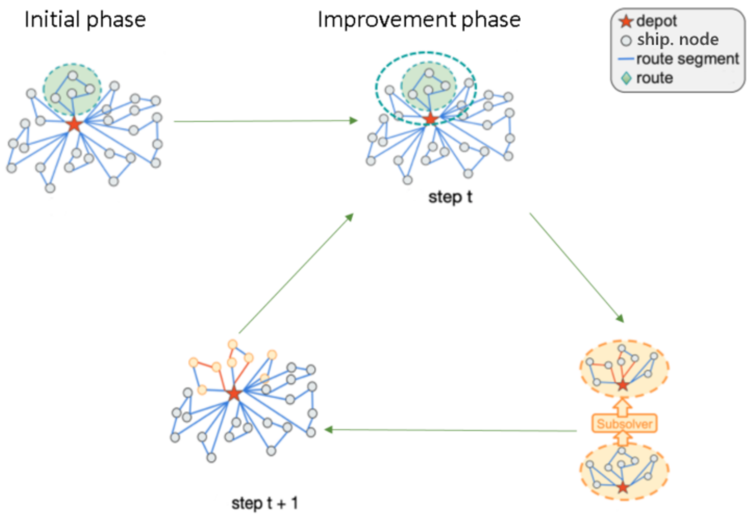


Figure 1.1: Image adapted from [27]. The optimization iteration starts by assuming initial phase (top-left), from which a neighborhood is chosen (top-right), optimized (bottom -right) and updates are added to the route assignments (bottom-left). That is the new initial phase for the next iteration, where the steps are repeated.

1.1.3 VRP as a business problem

This research project is done in the context of Dassault Systèmes' Logistics Planner. It solves a VRP with time windows and capacity, whose formal definition has a real world equivalent. Because of that, the values assumed so far are altered based on the method of representation.

The cost c_{ij} of travelling from vertex i to j is represented by the sum of business costs (distance, travelling time, etc.) and penalties for violation of business rules (delayed shipments, unmet requirements for the conditions of the shipment, waiting time, etc.). Within Logistics Planner this is termed **virtual cost** (also referred to as VC). **Optimization** of the problem is the act of decreasing the virtual cost of the problem by reassigning shipments between routes.

In LP, a depot usually has a *heterogeneous fleet*: multiple vehicles with varying properties and availability. A vehicle's capacity C is a function of is properties. Furthermore, a depot has a limited number of vehicles. As a result, one vehicle might need to execute multiple trips. Thus, a circuit c_k from the graph theory definition of a VRP is one **trip** of the vehicle and its **route** can consist of multiple trips.

1.2 Logistics Planner

The research is developed and designed to accommodate Dassault Systèmes' Logistics Planner and its optimizer. The optimizer is a tool for iterative improvement of a solution. It receives as input an initial route assignment of shipments provided by LP. The generated output is a potentially better solution that is supplied as input of the next optimization step. To clarify the role of those input and outputs in LP, the steps taken by the optimizer in order to solve the VRP will be listed here and depicted in Figure 1.1.

1. The optimizer receives the input of a VRP, consisting of a depot and shipments, as described in Section 1.1.

2. The first part of processing the input is the construction phase, where an initial assignment of all the shipments to routes is allocated. A good allocation can be attempted from the start, but no claims are made for how optimal it is. Here, an initial *virtual cost* is set, to be minimized throughout the next steps.

   Virtual cost is defined by business cost and penalties for violating business rules. Examples are:

   - **Business cost per route:** The tangible cost of running the route. This includes, but is not limited to:
     - A function of the distance covered by the route, as it has fuel costs;
     - The duration of the travel, including driving, loading and unloading times as the driver's time is also paid;
   - **Violation of business rule (per route):**
     - Going over capacity;
     - Routes exceeding the length of a driver's shift;
     - Waiting time;
   - **Violation of business rule (per shipment):**
     - Unplanned shipment: needs to be a steep penalty, as an unplanned shipment has no business costs and otherwise there would be no incentive to plan it.;
     - Delayed shipment;
   - **Additional costs:** It is possible to add situational costs. Examples are: CO_2 emissions, battery depletion for electrical vehicles, etc.

   A deeper dive into those steps is not the topic of this research, and therefore they will only be treated as a black box with in- and output. While the current research project is considered in the context of LP, in terms of generalization it will be able to fit any VRP solver with a similar setup.

3. Next the improvement phase starts. LP has an optimizer algorithm to minimize that virtual cost.

   - A sub-problem consists of a limited number of shipments in proximity to each other. It takes a neighborhood as its input, which consists of the following.
     - An **anchor** is a shipment selected from the problem, that the sub-problem will be generated around. Anchors are generated via a heuristic with high probability for all shipments to eventually be included in a neighborhood (defined in the next point).
     - A **neighborhood** is generated from an anchor, by selecting the closest shipments (note that here "close" is in terms of both physical distance and time for delivery). When a shipment is added to a neighborhood, so is the entire route it is on. Thus, a neighborhood consists of one or more complete routes.

     This research project does not assume a database of pre-made neighborhoods, because they can overlap and vary in sizes, which would result in them being exponential in number. Instead, they are generated as needed, via an anchor.
   - A neighborhood is then run through the optimization algorithm as a sub-problem. While the optimizer works on a neighborhood, it does not consider anything outside of it.

- The solution of each sub-problem is directly integrated in the original global VRP: an optimized sub-problem consists of a new assignment of its shipments to routes, which can be inserted in the main problem, re-assigning only the affected shipments. Thus, a better solution of a sub-problem improves the global result after each iteration.

4. Optimization is then re-iterated for different neighborhoods until interrupted. After each iteration, the VRP has a usable solution.
5. As defined in Section 1.1, if the virtual cost of the VRP is minimal, the problem is considered optimized, and the achieved solution optimal. The iterative sub-optimization approach does not guarantee an optimal solution, but after a number of iterations, the virtual cost of the problem will converge to a (potentially local) minimum.

The number of iterations until converging and the proximity of the convergence value to the one of the optimal solution is widely used as criteria for performance of the optimization algorithm. In this research the performance will be evaluated based on the virtual cost after a fixed number of iterations and not after a fixed duration. That is done because the method is not necessarily optimally implemented.

1.2.1 Neighborhood selection

While there are multiple ways to implement the optimizer, most of them are time and resource intensive and the one used by Dassault Systèmes DELMIA is not an exception. The goal of this research will not be to change the optimizer, but instead to select promising VRP regions for the optimizer to focus on. Phrased differently, the goal is to perform selection of the neighborhood for optimization.

While LNS generates neighborhoods of various solutions that differentiate only on a region of the VRP, in a neighborhood selection of interest, only the region is subject to change. Given an anchor, the rules used to generate the neighborhood are set by the LP (Figure 1.2 (c)). An anchor alone does not provide enough information for an entire neighborhood - it is just a shipment, selected to further find other nearby shipments. Furthermore, with the methods used by LP to select an anchor, it is possible, but not guaranteed, that the same anchor is selected more than once throughout the optimization - and hence the same neighborhood is generated more than once. The assumption that from some point on, all neighborhoods will be encountered and only repetitions will occur, is thus not reliable. Additionally, even when encountering the same neighborhood a second time, its optimization capacity will be different from the first encounter, as it will already have been optimized once.

That reasoning leads to the goal of this research project - once a neighborhood is generated, to decide if it is to be optimized or not.

Before optimizing a neighborhood, it is difficult to estimate what its **optimization capacity** is. This would require making a heuristic that, given a neighborhood, approximates how much optimizing reduces the virtual cost. It is difficult to achieve it, because the scale and properties of neighborhoods are vastly different and the optimization capacity of a neighborhood depends on the combination of routes in it. For example, two neighborhoods that overlap in all but one route could have vastly different optimization capacity, because the differentiation route is underutilized in one of them. Instead of a defining such a heuristic, a number of neighborhoods will be generated (10 in the scope of this project), compared in terms of optimization capacity, and the best one for optimization will be selected. Then, even in a borderline case, small but influential differences between neighborhoods can be observed in the context of the conditions in question.

In the next Section 1.3, the neighbourhood selection problem and its parameters are detailed. Subsequently, Section 1.4 provides a literature review on the topic, followed by the steps towards the solution in Section 1.5.

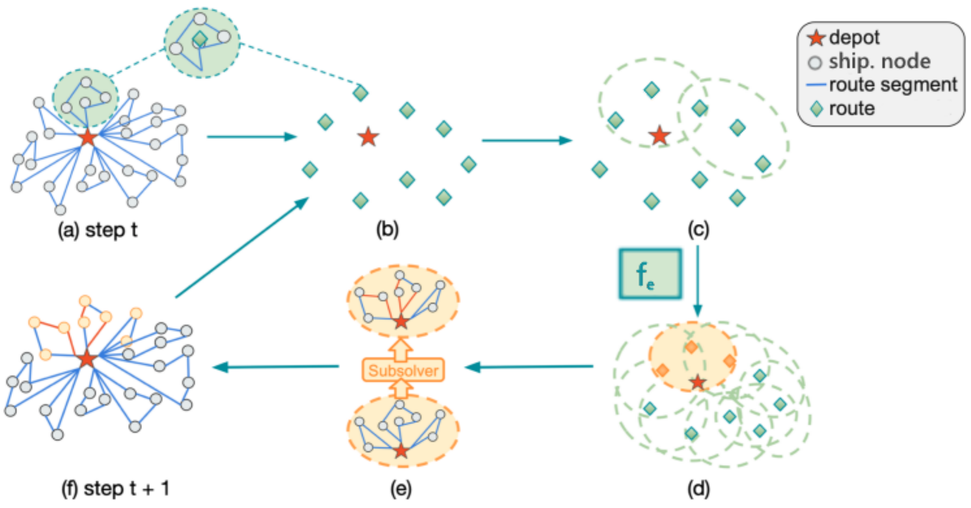


Figure 1.2: Image adapted from [27].
a) Each time step starts with a current solution X. Circles represent shipment nodes, blue lines stand for the path a vehicle takes between two shipments, and the red star marks the depot.
b) We represent a route as a point (diamond) for simplicity. Note: all routes contain the depot (at least twice).
c) Selecting neighborhoods (can vary in size and how much they overlap).
d) This project: selecting a subproblem S (yellow).
e) We feed S into the sub - optimizer to get a new sub-solution X'S. The red edges are updated by the sub - optimizer.
f) We update X to new solution X0 with X'S, then repeat (b)-(f). The solution achieved at f is the new (a).

Optimization capacity of a neighborhood

Assume the solution of the global VRP has a total virtual cost VC_{t-1}, before the optimization of the neighborhood selected at the t-th iteration. After optimizing the selected neighborhood, at iteration t, the total virtual cost is VC_t. $VC_{t-1} - VC_t$ is presumably a non-negative number. If it was not, the change would not be applied, making the improvement zero. The goal is, therefore, for the neighborhood optimized at time $t-1$ to be such that $VC_{t-1} - VC_t$ is maximized. In other words, the aim is for a neighborhood optimization to result in the biggest reduction in virtual cost of the overall problem. Within this research project, the value of $VC_{t-1} - VC_t$ will be called the **optimization capacity** of the neighborhood selected at time t.

1.3 Problem description

This section defines the problem the current research project aims to solve. To summarize: given a (time-costly) VRP optimizer, it is preferable to use it only on neighborhoods where it will yield a greater reduction of virtual cost. Such neighborhoods need to be identified.

This can be achieved by first assuming a set of neighborhoods (e.g., in Figure 1.2 (c)), and then deciding which of them has the optimization capacity (Figure 1.2 (d)). A neighborhood consists of multiple shipments, distributed over one or multiple complete routes and may overlap with other neighborhoods. For the purpose of this project, 10 neighborhoods are provided, but the number and their size may vary.

1.3.1 Goals

The goal of this project is to use GNNs to identify which of these neighborhoods would benefit the most from being processed by a VRP optimization algorithm: In other words, estimate the neighbourhood's optimization capacity (f_e from Figure 1.2) in comparison to other neighborhoods. This is done in order to find the best neighborhood relatively fast and pass it to the optimizer to only run on one sub-problem per selection, instead of on all the candidates or an arbitrary one.

Success in achieving this goal will be measured in terms of optimizer performance when using the hereby developed neighbourhood selection method, versus its performance without it.

Performance of an algorithm for real-time problem optimization, is usually measured as the achieved optimization for a specific runtime, but those are dependent on efficient implementation and consistent hardware. For comparison purposes within this project **performance** is defined as value reduction of overall virtual cost after a consistent number of iterations, which can be compared with other methods used so far by DELMIA. A performance measure is thus taken after applying the entire optimization algorithm - initiation, neighbourhood selection and sub-problem optimization - for a certain amount of iterations.

Therefore, the goal here is to achieve better performance - as defined above - than with previous approaches taken by DELMIA.

1.3.2 Proposed method

The method proposed by this project will use graph representation of neighborhoods. As a VRP has inherent representation in graph theory, and is a Combinatorial Optimisation Problem of the type "points to visit on a path", it is intuitive to address its sub-problems with a graph structure as well [25]. Further argumentation is sourced from literature in Section 1.4.

The intention is to use the graph representation of the data to train a GNN that can learn the **embeddings** of a neighborhood. These are learned properties of the graph representation that a further model can use to compare the neighborhoods the graphs represent. For this, a GNN leverages both the relationships and features of the neighborhood's graph representation. The embeddings are then used to predict its relative optimization capacity.

The GNN's goal is to provide the embedding of a single neighborhood. Then another model, a binary classifier, selects the one out of two neighborhoods, which has the greatest optimization capacity. All n neighborhoods are compared pairwise with one another. The neighborhood classified as having the better optimization capacity most times of the $\frac{n(n-1)}{2}$ comparisons will be selected for optimization. In the current applications, $n = 10$ and consequently the pairs are 45.

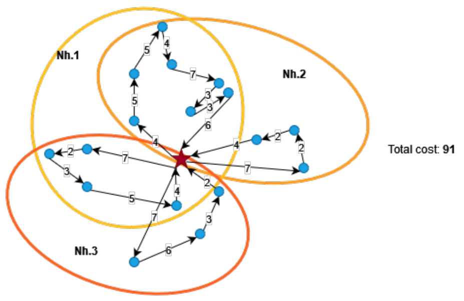
Because VRP is based in graph theory, as described in Section [11] it and its solutions and planning details can be represented in graph shape in an informative manner. Thus, GNN can directly take such a graph representation as input. This makes GNNs preferable to using any other neural network (NN), like a Convolutional NN to learn the properties of a sub-problem of VRP.

By definition, a GNN outputs an embedded graph of the same shape (in terms of nodes and edges between them) as the input one. Hence, its generated neighborhoods embeddings will contain information on both the features (represented as nodes) and relations (represented as edges) of shipments.

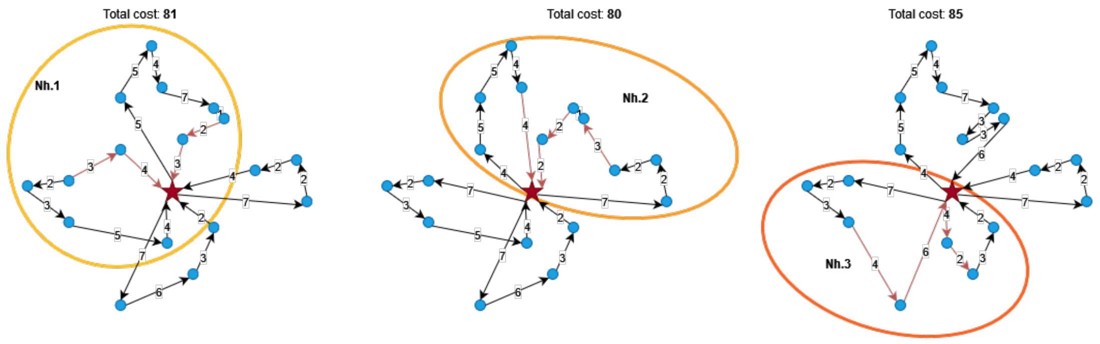
1.3.3 Example case

To illustrate the problem, assume a set of shipments already assigned to a route, as described in Figure 1.3a. Without loss of generality, assume routes and trips overlap (thus a route consists of a single trip). All routes originate from the depot and the edges are directed and weighted with the virtual cost of travelling from one point to the next. Note that the cost is not necessarily the length of the respective edge - it is instead the virtual cost, defined in Section 1.1.3. The total virtual cost of the problem is the sum of the weights of all the edges.

Assume further, that $Nh.1, Nh.2$ and $Nh.3$ (yellow, orange and red respectively in Figure 1.3) are 3 neighborhoods selected as optimization candidates. In Figure 1.3b the result of optimizing each of the neighborhoods is shown, along with the total virtual cost of the problem after each respective optimization. Note that the optimization takes features into account that are not shown for simplicity, like time windows.



(a) Example representation of shipments and a depot as a graph, with added neighborhoods to select from and the total cost of the problem. The location coordinates of a node represent the location of the corresponding shipment and the weights of edges - the virtual cost of the trip from one shipment to another



(b) Example results of optimizing each of the neighborhoods

Figure 1.3: Example problem

The goal of this research project is to create a machine learning (ML) model that is capable of recognizing that optimizing $Nh.2$ provides the greatest total virtual cost reduction within the scope of Figure 1.3 without optimizing all the neighborhoods.

Note that in 1.3, $Nh.2$ and $Nh.3$ have no overlap and if both were to be optimized, an even better solution would be found. The example goes through a single iteration of neighborhood selection.

1.3.4 Data description

Prior to the actual decision-making, the GNN needs to be trained on appropriate data. A pre-made dataset, populated with neighborhoods and their labels, is not available. As mentioned before, the neighborhoods are generated dynamically on each optimization iteration: An anchor is selected and a neighborhoods is generated around it. Then the sub-problem is processed through the optimizer and finally replaced with its optimized version within the overall problem. The only obtainable optimization capacity of the neighborhoods is the total virtual cost of the problem before and after optimizing it.

Therefore, the training dataset is generated by recording the neighborhoods as they are created during a run of LP's optimizing algorithm. The labels are defined as the virtual cost improvements after each neighborhood is optimized. Note that each iteration will be optimizing the global problem further, and could eventually converge to a (locally) minimal-cost solution. Therefore a neighborhood optimized later in the solution process will likely yield a smaller improvement than the same neighborhood being optimized early on. Thus, label comparison needs to be limited to similar stages of optimization.

1.3.5 Research Questions

This project presents the following research questions:

First, what is a good graph representation of a neighborhood from the Vehicle Routing Problem (VRP), to train a Graph Neural Network (GNN) in order to compare different neighborhoods in terms of how much optimizing that neighborhood will reduce the total virtual cost of the VRP (optimization capacity)? Answering this requires an initial decision of a good single neighborhood graph representation, followed by training a GNN for this comparison in order to evaluate the representation's quality.

Second, how to compare graph representations of the neighborhoods in terms of optimization capacity? Graph representations are expected to be initially processed individually via a GNN, which poses the question: What method of pairwise comparison is needed for neighborhoods represented as GNN-embeddings to decide which one has the bigger virtual cost reduction when optimized?

Third, does a GNN provide additional value in terms of virtual cost minimization after a number of iterations when compared to other ML methods, if used for selecting a neighborhood with the greatest improvement in each iteration of the optimization, from a set of neighborhoods? Specifically, graph based neighborhood selection will be compared to another ML based neighbourhood selection method and iterative solving of sub problems as described in 1.2. Additionally, does a transformer or a GNN perform better in the same terms?

1.4 Literature Review

This section will present an overview of the use of ML, and more specifically GNN, to solve VRP using Large Neighbourhood Search. Its goal is to put the current research into the perspective of methods and techniques used in the area and to showcase where neighborhood selection fits in VRP's solutions.

LNS for VRP has been researched for more than 20 years, from purely mathematical approaches [34] to state-of-the-art ML-based methods [32], [20]. In many of the recent approaches, GNNs for LNS are used in order to find a (near) optimal solution for a VRP. This literature review will relate them to the neighborhood selection applied in the current project, in terms of directly described methods, or applicable ideas.

Some main approaches taken towards structuring and solving VRP via neighborhood selection using GNNs in literature will be discussed, detailing the tools used. The review will also branch out into combinations of those three topics, with more relaxed parameters, focusing on their applications to current research. In particular, first the general framework for approaching the problem will be described, followed by setting up the basics for Destroy and Repair method; encoding and how it can be a way to make graph representation and heuristic estimations. The review will conclude with attention mechanisms and alternative approaches.

1.4.1 Framework: Representation of LNS for Optimization

There are a few ways to approach the problem in this project. The method used by the LP in Dassault Systèmes' practise is highly reminiscent to the one in [27], from where Figures 1.1 and 1.2 are adapted. There, however, neighborhood selection starts by taking the centroids of routes and then - a neighborhood of n centroids.

Afterwards, [27] uses the properties of the centroids instead of a graph representation of neighborhoods and proceeds similarly to the approach in this project. It estimates the same optimization capacity, as defined above, compares it to other neighborhoods (of centroids), and selects the best one. Finally, a transformer is used to optimize the chosen neighborhood to reassign necessary shipments to routes such that the reduction of virtual cost, estimated at selection, can be achieved. The authors of [27] report 1.5 to 2 times improvement in conclusion speed over other methods - specifically Lin-Kernighan-Helsgaun and variations - in problems with up to 3000 shipments.

Note, that [27] does not explicitly mention to have used GNNs. It can be observed that there is little difference between a transformer and a GNN. A transformer is, in functionality, a subtype of GNN, where the input is a fully connected graph. It is more limited than a GNN, as claimed by [10]: "Such architecture does not leverage the graph connectivity inductive bias, and can perform poorly when the graph topology is important and has not been encoded into the node features". Meaning it assumes strong connections between all nodes, regardless of necessity. In turn, a GNN tackles edges that may or may not exist, and both leverages a graph's topology and, according to [42], can have a transformer layer if needed, when all the edges are assumed to be present for it.

1.4.2 Method: Destroy and Repair

The most common approach to LNS is the partial destruction of a solution (de-assignment of shipments from routes within a limited neighborhood) followed by its repair (reassignment of shipments in potentially different routes and order).

Before proceeding further, it is important to note that [4], as well as various other works ([35], [38], [13], which are described in further details in the next Subsections) define a neighborhood in LNS differently from this project. By their definition, LNS builds a neighborhood of whole solutions of VRP with minor differences between each other. Those neighborhoods of solutions are generated within LNS by the Destroy and Repair operators. The former partially destroys a solution within a range around an anchor (e.g. set by a limitation of the number of nodes to be destroyed). This is then repaired by the latter, using varying assignments of shipments to routes. This creates a number of slightly different solutions, which combined are a **neighborhood of solutions**. The limited range of the destroy operator prevents the repair from re-calculating the entire problem from scratch. Thus, the solutions in a neighborhood are the same for the most part, except for the range of the destroy operator (e.g. the marked one in step (a) in Figure 1.2), where their assignment to a route differs (the solutions in step (a) and (f) in Figure 1.2 would belong to the same neighborhood).

In that representation of neighborhoods of solutions, variations occur solely within the destroyed set of nodes. This can be perceived as a range around an anchor. Thus the area, where the reassigning occurs can be considered a selected neighborhood- sub-problem that is optimized, rather than one of the possible solutions of LNS. Unless stated otherwise, in this project, a neighborhood will be considered to be a subset of shipments in a solution.

In the following sections it will become evident that the Destroy and Repair method is a core principle of LNS, as many of the discussed approaches to LNS either perfect, expand or alter it.

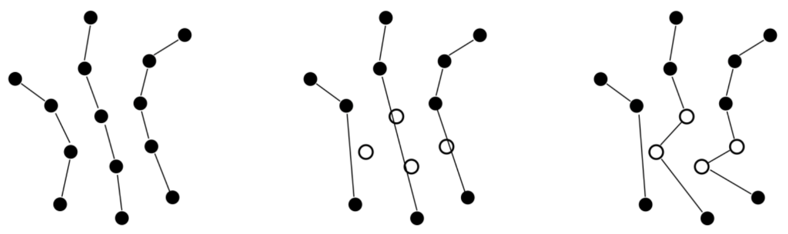


Figure 1.4: Image originally from [4]; An example of large neighborhood search. From left to right: the original solution; a destroyed one; a repaired one.

1.4.3 Method: Encoder as Embedding

The other large class of approaches is to encode a neighborhood (be it a neighborhood of solutions, from literature, or a sub-part of VRP, as per this project) to a function of its properties. It is then used to calculate how to minimize the value of the encoded representations. Afterwards, it is decoded back either to the neighborhood (if required e.g. in [38]), or to a type of heuristics[17], or another representation as needed. While the encoding can have various forms, the focus here will be encoding towards an input for GNN - a graph representation.

In [17], the encoding is done by first representing neighborhoods as graphs and then integrating them further with a Graph Attention Network (GAT). The graph representations initially contain nodes described via demands (shipment and dependent corresponding route information) and edges depicting the topology between all nodes. This representation is directly applicable to the current research. Importantly, in [17], the decoder is not used to try and deduce a solution, as some other encoder-decoder methods do, but instead learns a heuristic for each node - used specifically to guide the LNS destroy operator as to which node to destroy, so that it produces a subset of nodes to be removed and re-assigned to routes that would provide the greatest cost improvement. On a VRP instance of more than 400 nodes, the approach converges much faster than other models - ranging from random heuristics, to runtime adaptive ones such as Slack Induction by String Removals [5] - and achieves a closer result to the theoretical ”perfect” solution than these other methods.

The only necessary alteration to the method in [17] in order to apply it to the current research is that the learning would not be with the goal of selecting each element of the set to be re-optimized, but selecting one of two pre-defined regions. This is a simplification of the problem described in [17] and should lead to even better performance.

As depicted by [38], generally neighborhoods get encoded to receive informative embeddings for a GNN (or another type of NN) to be trained on. In [38] specifically, the neighborhoods are further encoded via a Variational Recurrent GNN [20] to be graph embeddings. To do that, the entire VRP is modelled as a variational probability problem with prior and posterior probabilities of route assignment. Then, a variational graph message network - Graph Reasoning Network [39] - is applied to the model. The authors proceed to decode the so encoded neighborhoods via a genetic-adjacent algorithm (which is outside the scope of this research). The method tends to converge faster and has its initial results as soon as it starts, rather than only being usable after training. On problems of up to 100 shipments, the result is up to twice as good as other methods tested on in [38] on early iterations. While the variational approach is also beyond the scope of this project, the concept of encoding the initial problem with a graph representation, followed by a GNN training is not, and ideas can be gained from the method used in [38].

Such an approach can be further developed, as [13] shows, by stacking several GNN layers to increase the spatial resolution of the model - and hence the maximum size of the neighborhoods - or adding a second GNN to create dynamic embeddings - e.g., if the initial graph representation does not feature edge directionality, but it needs to be learned at training.

1.4.4 Method: Heuristics for a neighborhood learned via GNN

Another method to approach LNS in the context of VRP is to make a heuristic that encapsulates the improvement capacity of a neighborhood, given specific features. To calculate such a heuristic, information needs to be embedded in the format required by the approach - in the case of the current research, this is graph embedding. Papers [4], [34] and [35] feature such a representation of the information, which can be used in this project's graph design. Furthermore, the way those heuristics are used can be a source of inspiration on how to use the graph representation in the GNN that this research will leverage. Additionally, a sub-heuristic can be used, estimating the relevance of different features to leverage in the representation.

This research paper will use a combination of the various heuristics described in this section. Such an approach was first coined as Adaptive LNS in 2006 in [34], where it is explored only mathematically, whereas this project will mainly consider heuristics inferred by ML. This method can be combined with others, as in [17], described in 1.4.3.

Dynamic partial removal

The above discussed methods of Destroy and Repair (Subsection 1.4.2) and heuristics (Subsection 1.4.4) are combined in [4]. There, a neighborhood of solutions, as defined in Subsection 1.4.2, is used. This neighborhood is generated via the Destroy and Repair method, where the destruction is done using heuristics from Subsection 1.4.4. Using heuristics to decide what sub-problem of the VRP to consider is very close to the method used in LP (Section 1.2). The paper discussed here is hence one of the most influential ones in the current research.

As discussed, the Destroy operator starts from an initial solution, deassigns some of the shipments from their routes, and then the Repair operator binds them back in slightly varying configurations, as shown in Figure 1.4. In [4], specifically, the Destroy methods uses a heuristic, calculated via dynamic partial removal - it iteratively completely removes nodes from the graph representation. It assumes the shipment represented by a node does not exist and the route it is on just skips from the previous to the following shipment. The heuristic then estimates how much the virtual cost change without the node. The process is repeated for each of the nodes surrounding the original node, both on the same route and adjacent ones. Afterwards, a pre-set number of nodes, whose removal reduces the virtual cost the most, are actually removed by the destroy operator. Compared to other heuristic solutions, dynamic partial removal as applied in [4] achieves faster convergence and slightly better solutions within consistent time on VRP problems with up to 200 nodes. Both the general idea of heuristic classification and the specific selection of where to start discussed in [4] are directly applicable to the current research.

The anchor node for the removal is selected in [4] by a Hierarchical Recurrent Graph Convolutional Network, trained on embeddings of a graph representation of solutions: with nodes defined by the shipment's location, time window etc. and edges as the temporal and spatial relationship between the nodes. The edges are inferred by the network, and within the scope of this project, the same could be done to a neighborhood- in terms of suggestions whether an edge should be there or not. While in the research in progress neighborhoods are already defined by LP, (the nodes to be removed cannot be selected one by one), the used graph representation can be fitting because it gives information on how many and which of the nodes in it need to be re-assigned to different routes. Additionally, while [4] converts the GNN embeddings to solutions, they can instead be used to calculate the distance between suggestions made by the GNN and initial edges as a heuristic for the neighborhood's optimization capacity, similar to the method in [17].

Meta-heuristic

Another use of heuristics for LNS is in meta terms. Paper [35] considers the solution of a specific VRP as a variation of a general solution, where specific paths are to be generated, given a problem and parameters that dictate how to solve it. Then, the complexity comes from the necessity to define the parameters of the problem - such as rules to adhere to, weight of features, etc. - manually. That is usually done by running multiple simulations with different parameters and selecting the best performing ones. To mitigate that difficulty, [35] discusses training a neural network to predict these parameters instead. While such parametrization is overall avoided in the problem description of this project, it shows that various features, based on the problem instance and its graph representation's statistics, can be used to predict the solution quality of a problem, before that solution is generated. That could be useful in the current project, if the same information is taken for a neighborhood. The approach proposed in [35] managed to predict the virtual cost that optimization will eventually achieve with 95% accuracy, for problems with 2500 shipments and 700 vehicles. While applicable on VRP, the experiment was conducted on the Ride Hailing Problem, which, unlike VRP has a separate origin for each destination. The current project will not deal with the parametrization but applies the key concept, the determination of which features of a neighborhood are most important for its optimization capacity.

1.4.5 Method: Attention

A Neural Large Neighbourhood Search (NLNS) approach is proposed in [21], mentioning the concept of "good enough" heuristics - those that provide sufficient directions as to where the solution can be improved. "Sufficient" here is measured in terms of quality of the eventually achieved solution. The authors use a deep network with an attention mechanism.

Attention mechanisms, reviewed in [33], while varying in complexity, can be described as a table that matches each node with a list of the strength of its relations to other nodes. The same thing is achieved in a GNN, if the edges and their weights are defined to represent the same. Therefore, it is beneficial to mention methods that leverage an attention mechanism, as it can be incorporated in the architecture for the GNN used to solve the current problem. An attention layer can be added to a GNN, while keeping the input edges available to store other information, as described in [37], making it a Graph Attention Network.

In [21] the results of a comparison between a base neural network and one with an attention mechanism added can be seen. The authors use the above-mentioned Destroy and Repair technique on LNS and have all the embeddings of destroyed neighborhoods (chosen here via randomization) run through a neural network with an attention layer to guide the search. The proposed method in [21] significantly outperforms ones lacking attention on instances with up to 100 shipments.

The authors develop NLNS further in [22]. They specifically contrast GAT-s, like the one in [7], which generates solutions for the problem via a gradient, with their method - using multiple attention layers to distinguish. They henceforth describe the relevant embeddings and show convincing results of near-perfect solutions in problems of 100 to 200 nodes.

Both [21] and [22] proceed to use NN in the form of integrated embeddings for Repair function. This project instead uses the so learned heuristics similarly to the handcrafted one in [27] to identify promising sub-problems with higher precision.

Single player game with attention and without LNS

Approaching VRP in a way that does not use LNS requires changing the point of view - like viewing the problem as a single player game. While those methods do not have a lot to do with neighborhood definition or selection, they still require a graph representation of the problem - which can be useful in the current project, by selecting a sub-graph as a neighborhood. This subsection will further consider how to do the eventual comparison of two neighborhood embeddings.

The authors of [28], do just that - treating VRP as a single player game and assume an agent seeking to generate a solution: iteratively taking in the state of the system and undertaking an action based on it. That still requires a graph representation of the problem with nodes - location, time window and demands - and edges between each pair of vertices - the distance between the nodes. In [28], a neighborhood (as defined here - a sub-problem of VRP) is built by an attention model. It takes the current state - the graph of the neighborhood already built and accessible nodes as neighbours as input to potentially include. Consequently it evaluates the probability of visitation - dependent on how much value those nodes will add if included in the neighborhood when it is optimized. Here, visitation of a node means taking it into account when optimizing the sub-problem. Then, decoding is done with actual visitations. The approach finds an acceptable, even if not optimal, solution (with gaps from an ideal solution of 20% to 40%, depending on the sampling approach) fairly fast. It is easy to expand to similar problems, and is a source of inspiration on the graph representation.

This approach is generalized in [9] to the general class of Combinatorial Optimization Problem on graphs. This can be done by taking the problem's graph representation, whose shape is dependent on the specific subclass of problem, as an assumption. With it, [9] proposes a GNN-based method that can find a solution for the problem in linear time once the GNN is trained. By achieving such low complexity, [9] shows that Combinatorial Optimization Problems and their sub-problems, such as the one considered in the current research, are indeed a good fit for graph structures.

Another expansion is offered by [25], where the load on the attention mechanism is reduced by creating the initial graph representation with constraint satisfiability in mind - making sure to satisfy as many constraints as possible from the start. The authors of [25] use more elaborate edges for graph representation, that encode both the relationship of a node with its immediate neighbours, and subsequent relationships with further nodes. This approach allows for better decision-making and is of greater interest when a neighborhood is generated.

Furthermore, [25] uses a Structure2Vec approach - recursively defined network architecture, inspired by graphical model inference algorithms - to obtain the final solution from the graph representation. A similar technique can be used in the final step of the current project, when the GNN embeddings need to be compared in terms of a criterion not necessarily part of the embedding (in this case - the virtual cost of the whole problem before and after optimization). The Structure2Vec approach is shown in [6] to quickly learn such comparisons using a much smaller model than the ones otherwise required: comparison of embeddings with kernel methods.

1.4.6 Literature review: Conclusion

This literature review focuses on papers that, in one or more of their taken steps, are applicable to this project. Paper [27] depicts the framework for this project and [17] lists a close representation of the steps this project will take, making them more interesting, but useful elements for this project are found in all the mentioned papers. Most of them use various graph representations, a combination of which will represent the neighborhoods in this project. Additionally, they explore the usability of different types of GNNs and their relationship with transformers and attention mechanisms. Finally, inspiration may be gained as to how to compare neighborhood embeddings.

1.5 Plan of approach

To answer the research questions, this project will take a number of steps.

First, one or more graph representations of a neighborhood need to be prepared for testing. The representation will serve as input for a GNN and requires creating an initial graph design, then choosing which data features to encode in the graph (whether there should be global nodes, types of nodes and edges, etc.) and how to encode them. Finally, the type of graph needs to be inferred (whether direction of edges or full connectivity is necessary, for example).

Second, depending on the representation made, the graph neural network to embed the encoding will be selected. Next the GNN's architecture and loss function will be chosen and implemented, following by gathering and encoding the training data for the GNN graph representation. Its processing will likely overlap with the previous two steps. Finally, the GNN's model will be implemented.

Once the model is completed, trained, and has generated embeddings, these will undergo a pairwise comparison. A follow-up binary classification model will, given two embeddings, select the preferred one (using criteria labels, depicted in 1.3). This second binary classifier also will be designed and implemented. Next one of the 10 neighbourhoods will be selected for pairwise comparison in order to obtain results.

Finally, the implemented algorithm will be evaluated on simulation of the problem in real-world usage with a fixed number of iterations, then tested against other existing models and variations. Assuming multiple representations, they will be compared to each other and existing approaches via experimentation on the evaluation stage. Finally, the results will be used to answer the research questions from 1.3.5: What is the implemented model's performance, what graph representation benefits the GNN, and how to compare them.

Chapter 2

Graph Design

2.1 Graph representation

The first step to using a GNN for neighbourhood selection is representing the data as a graph to serve as input for the GNN. To design the representation, decisions need to be made on the graph's structure: types of nodes and edges, as well as which features are to be included and where.

In order to ensure only useful properties are included in the graph, an iterative approach to adding them is leveraged, where each new property is added, tested in terms of the value it provides to the overall performance, and kept for the next iteration only if it is beneficial. Some properties have multiple options, neither of which can be argumented to be better than the others prior to implementation. In these cases both will be experimented with to choose the better one.

This chapter describes that process of designing and building the graph representation, including the phases of its development, the approach taken towards decision-making and its argumentation, and the graph's type, structure and properties.

2.2 Iterative graph representation framework

This section outlines the iterative graph building process. Consequently the planned Graph Phases are listed, along with the argumentation as to why they are of interest.

The graph will be built with iteratively increasing complexity, in order to avoid working convoluted structure that is heavy to process, without evidence that (most of) its elements assist the models' learning. In order to achieve this, a linear approach is used. The developmental iterations will be referred to as **Graph Phases**. The first one only includes the basic information necessary to define the problem. As will be argumented in the next section, this consists of the shipments and depot as nodes, with the paths between them taken by the vehicles moving in the VRP, as edges. The Graph Phase will be evaluated in terms of performance in simulation of a real world problem, as described in Section 3.6. Evaluating this one and any following Graph Phases requires a basic pipeline to be set up: represent the neighborhoods as a graph, use them to train a GNN model for make pairwise comparisons between them, and then test it by using it in a real-world-like problem optimization process to repeatedly select the best neighborhood for optimization. Performance will be evaluated based on virtual cost reduction when testing in that scenario.

In each Graph Phase following the first one, an expansion of the graph will be added: a new node type along with typed edges between it and the previous nodes, extra connectivity between existing nodes, or extra information on existing graph elements. The improvements provided by the

		Representation
Nodes	Shipment Nodes	Deliveries vehicles need to make and the depot
	Trip Nodes	Trips vehicles make from the depot and back to make deliveries
	Route Nodes	Full routes (one or more trips) of vehicle throughout a day
Edges	Path Edges	The travel of a vehicle from one location to the next (between Shipment Nodes)
	Shipment-Trip Edges	Connecting Shipment Nodes to the Trip Node representing the trip those shipments are part of
	Trip Edges	Interconnectivity between Trip Nodes
	Trip-on-Route Edges	Connecting sequential Trip Nodes that are on the same route
	Trip-Route Edges	Connecting Trip Nodes to the Route Node representing the route those trips are part of
	Route Edges	Interconnectivity between Route Nodes

Table 2.1: The elements that will be added to the graph throughout the Graph Phase.

expansion will be evaluated and kept in the graph representation if they improve the performance. If the expansion is not shown to be useful and is not a crucial step for further ones it will be discarded. In case an expansion on which the next steps are heavily dependent reduces performance, its treatment will be considered on a case-by case basis.

2.3 Graph Phases

This section lists the phases of graph development in increasing complexity, along with argumentations as to why they are of interest. Most features assigned to the graph nodes and edges per Graph Phase are selected in accordance to the ones with the prior work on including ML in Dassault Systèmes' LP - using a Random Forest Classifier. Because of that, this paper will not detail all the features separately, but only the ones included specifically for the benefit of GNN usage. The general kind of features selected per graph element type is outlined for the purposes of reproducibility.

The expanding Graph Phases iteratively include the elements listed in Table 2.1. First Shipment Nodes are included, representing the deliveries vehicles need to make. Then Trip Nodes are added, matching the trips made by vehicles from the depot and back in order to make those deliveries and summarize them. Finally, Route Nodes are included, to represent the full routes (consisting of one or more trips) that the vehicles make throughout a day. With the introduction of each type of nodes, appropriate edges are created. The rest of this section will detail the properties and sequence of the elements added.

This method does not explore all possible graph element combinations. Excluded ones are considered unlikely to result in promising results. Furthermore, some Graph Phases described here are eventually not tested, based on the results in Section 4.

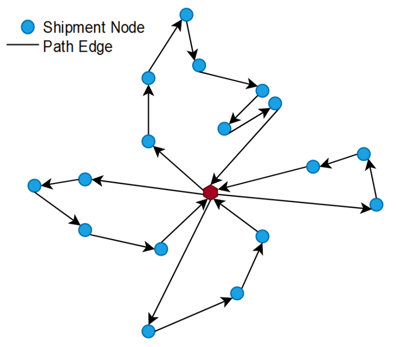
2.3.1 Phase 1: Just the basics - Shipment Nodes and Path Edges

		Representation
Nodes	Shipment Nodes	Deliveries vehicles need to make and the depot
Edges	Path Edges	The travel of a vehicle from one location to the next (between Shipment Nodes)

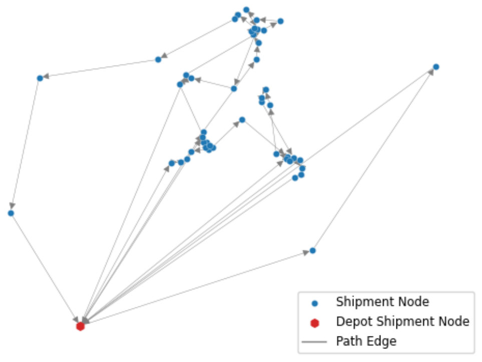
Table 2.2: The elements of the graph included in Phase 1.

The foundation of a VRP problem (and hence of the currently developed graph design) are shipments that need to be made (including pickups at the depot), and the path the vehicle takes from each visited shipment to the next (including the depot location). All further components - trips from and back to the depot, routes of vehicles during the day and their costs and properties, are based on these initial building elements.

Because of that, Graph Phase 1, as depicted in Figure 2.1, will only concern Shipment Nodes and Path Edges, noted in Table 2.2. It has no global information on trips and routes.



(a) Representation, as planned in advance.



(b) Representation of an actual neighborhood, taken from the data. Uses x-y coordinates of the nodes for positions. Note that, intuitively with the representation being of a neighborhood of routes, the Shipment Nodes are in similar positions with respect to the depot node.

Figure 2.1: Phase 1, with directed path edges, in black.

The Shipment Nodes are defined via a number of features of the shipment stops they represent - x, y coordinates, start, end, duration of the time window they are to be made in, and some properties of the current plan to execute them, such as the expected delay, per information from Dassault Systèmes' LP. Shipment Nodes and any other nodes that will be discussed have a unique index, used to define edges with respect to connecting nodes, but are not explicitly added as a node feature.

The depot is also represented as a Shipment Node, with its own coordinates, time window equivalent to the full time range of the VRP, and most of the plan properties - zeroed. This design decision is made due to the depot being the most similar, by its structure and purpose, to Shipment Nodes. If the depot was a different type of node, the edges connecting it to the first and last shipments on a trip would have to be their own type as well, since a type of edge only connects consistent types of nodes. Setting the depot as a Shipment Node, however, creates some circularity problems in message passing in GNN discussed further in this section.

The Path Edges only have a few attributes: Information about the planned solution of the VRP like distance and duration of the path the shipments. Reflexive edges from each Shipment Node to itself are added with matching attributes. These are necessary, in order to facilitate nodes' retaining own information via reflexive message passing. This holds for all types of nodes that will be introduced and reflexive are introduces with the mention of each new node type.

Phase 1 has two different options to be tested: The Path Edges can be one or bidirectional. In this Graph Phase, as well as all others with multiple variants, both options will be implemented and tested, and the one that performs better, as assessed via the method in Section 3.6, will be selected to build upon in following Graph Phases.

One directional path edges: Following the order of shipments

The first option for Phase 1 is, intuitively, to have the Path Edges be in the same direction as the vehicle movement, from a Shipment Node representing one delivery (or pickup), to a node representing the next delivery (or pickup). The arrows on the edges at Figure 2.1 are representative of that direction only.

An argument can be made for that representation, as the travelling sequence is in the same direction in time - the stop represented by a destination Shipment Node is by definition after at stop at the origin Shipment Node. Later actions cannot influence earlier ones, which is represented in the message passing direction.

Bidirectional path edges

The above approach does have some limitations. For example, while a delay indeed has no influence on the deliveries prior to it, if the plan was to change (as it likely will, if the neighborhood is selected and optimized), the order of the deliveries can be switched around. In the new distribution, the order of influence may thus be reversed and a heuristic for the results of re-planning needs to reflect that. A step towards this is message passing in the opposite direction of movement, as well as following it. Additionally, having graph cycles connected in the depot makes the latter a summary node with limited reach. Having bidirectional edges, makes the depot a summary of both the beginning and ends of trips, in contrast to only the ends. Given that bidirectional edges are represented as separate in both directions, an extra attribute is added to each, indicating if the edge matches the sequence of stops (the original on-directional edge), in order to preserve that information.

As will be discussed later, the GNN implemented here would be fairly shallow, with fewer than 10 layers. Conversely, a trip a vehicle makes between two visits of the depot could be much longer than that. As a result, when using only the direction of the sequences of visits, the last stops of a lengthy trip, will not be influenced by the first ones if the number of edges between them is more than the number of layers of the GNN. That is, because each GNN layer passes messages over just one hop - from each node, just to its direct neighbours.

2.3.2 Phase 2: Trip Nodes

In Graph Phase 2 Trip Nodes (with reflexive edges) are added, together with edges between them and the Shipment Nodes, as shown in Table 2.3. Their dependency is depicted in Figure 2.2. From this Phase on, the graph designs are heterogeneous. Because of that most experimentations with

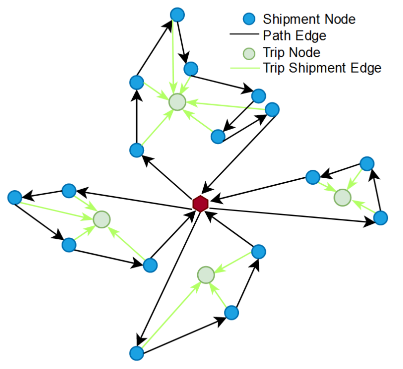
		Representation
Nodes	Trip Nodes	Trips vehicles make from the depot and back to make deliveries
Edges	Shipment-Trip Edges	Connecting Shipment Nodes to the Trip Node representing the trip those shipments are part of

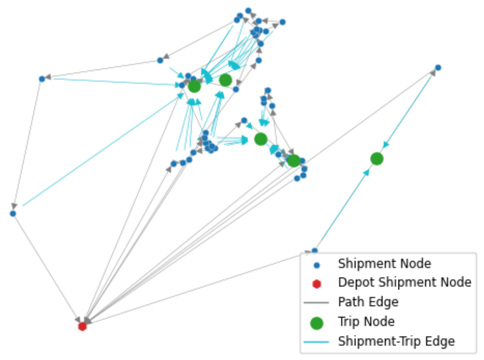
Table 2.3: The elements of the graph included in Phase 2.

GNN and tuning in order to handle heterogeneous graphs will be done in this Graph Phase, and described in Section 3.5.

Trip Nodes' features are a summary of the Shipment Nodes corresponding to shipments on the trip. Their coordinates are averaged over the respective Shipment Nodes, making Trip Nodes centroids of their Shipment Nodes. Included are features describing the trip's time window by using the earliest start from the shipments and the latest end. The total duration, cost and properties of the vehicle used during the trip are also added.

The edges between Shipments and Trip Nodes, carry an attribute of straight line distance, calculated based on coordinates. This can be used to infer how remote a Shipment Node is with respect to the rest of the trip. This Phase has two variants of implementations, with those edges being one or bidirectional, which are discussed below. For most of the upcoming Graph Phases, the additions included will only be kept if they provide better performance as per Section 3.6. However, this and the previous Graph Phases are fundamental to any following development of the graph and henceforth one of the options for them is guaranteed to be included at further development.





(a) Representation, as planned to advance.

(b) Representation of an actual neighborhood, taken from the data. Uses x-y coordinates of the nodes for positions.

Figure 2.2: Phase 2, building on Phase 1 by adding Trip Nodes and edges between them and Shipment Nodes.

One directional Shipment - Trip edges

Within real world interpretation, a trip is a sequence of stops made by a vehicle between depot visits. It is intuitive therefore for the stop-representing Shipment Nodes to have direct influence on their respective Trip Nodes. Within the GNN's graph, that is done via a directional edge from Shipment Nodes to Trip Nodes as shown in Figure 2.2.

Within the GNN, if there were no edges in this direction, Trip Nodes would have no incoming edges for this Graph Phase. Given that a GNN passes messages in the directions of the edges, one directional edges would mean that each Trip Node only gets messages from itself, which is incosistent with the real-world dependency of trips on shipments.

Bidirectional Shipment-Trip edges

If the edges are kept in one direction, as described above, it can be argued that Shipment Nodes should stay uninfluenced by the Trip Nodes. However, at the current Graph Phase, that would limit the influence of each trip only to itself. Additionally, even with bidirectional Path Edges, not all Shipment Nodes can pass messages to each other. Taking a message from the summary Trip Node would allow for each Shipment Node to receive that missing information. These messages can be passed by adding edges from Trip Nodes to Shipment Nodes. Assuming existent Shipment-Trip Edges, this makes a bidirectional relation.

2.3.3 Phase 3: Edges between Trips

		Representation
Nodes	-	-
Edges	Trip Edges	Interconnectivity between Trip Nodes

Table 2.4: The elements of the graph included in Phase 3.

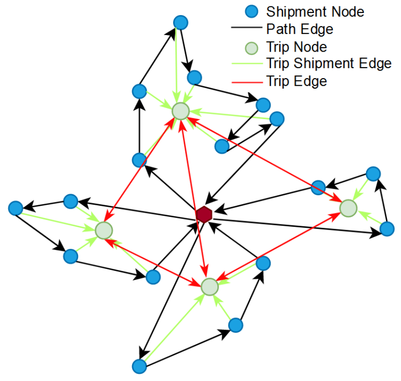
A lot of the limitations in Phase 2 can be overcome by adding edges between the Trip Nodes, to allow message passing between trips, as referred to in Table 2.4. There is no clear relation between trips of different routes, except the possibility they could influence each other, at optimization by moving shipments from one trip to another. Therefore, only bidirectional edges are relevant to evaluate connecting all Trip Nodes, as shown in Figure 2.3. An exception is the directional relation between trips in terms of sequence in time, which is explored in Graph Phase 4.

In this Graph Phase, the edges between trips only have a few attributes. These are whether they are on the same route or not, the absolute time between different trips' time windows and the distance between their coordinate positions. The idea behind those attributes is to make it easier to learn if parts of trips can easily be moved to other trips. The dummy-valued reflexive edges added for each Trip Node when they were introduced, are replaced with reflexive Trip Edges containing the appropriate attributes.

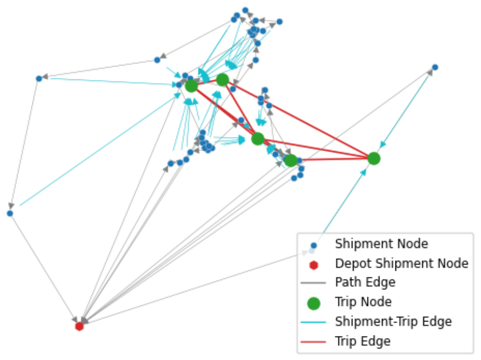
2.3.4 Phase 4: Sequences of Trips

One vehicle can make one or more trips within a day of operation. The route of a vehicle is the sequence of all the trips it makes within a single VRP. Often a route consists of only one trip, but when that is not the case the sequence of trips is interesting to represent. In this research project, two ways for that will be experimented with: adding a separate type of edge, as shown in Figure 2.4, and only adding attributes to the edges from Phase 3 (Figure 2.3), as outlined in Table 2.5.

Regardless of the method chosen, the edges between trips are altered to include information on time between trips. The change makes use of the fact that a bidirectional edge is represented by two single directional edges. In Phase 3 the time between trips is already included as absolute value.



(a) Representation, as planned in advance.



(b) Representation of an actual neighborhood, taken from the data. Uses x-y coordinates of the nodes for positions. Bidirectional edges are represented with no arrows on this plot, for readability purposes.

Figure 2.3: Phase 3, expanding on Phase 2 by adding edges between Trip Nodes.

		Representation
Nodes	-	-
Edges	Trip Edges	Interconnectivity between Trip Nodes
	Trip-on-Route Edges	Connecting sequential Trip Nodes that are on the same route

Table 2.5: The elements of the graph included or altered in Phase 4.

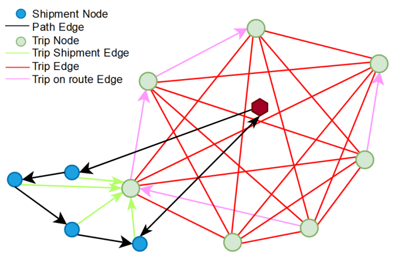
Conversely, here for each one directional edge, that attribute is kept positive or made negative, depending on weather the trip represented by the destination node of the Trip Edge is after the one on the origin node or not.

Trip-on-Route edges

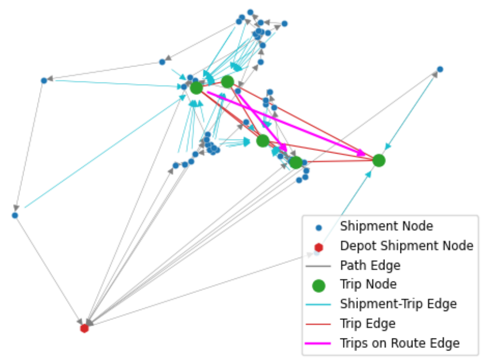
The first explored option for conveying a sequence of trips in a route is to add a separate type of edges between trips that are on the same route following the direction of the sequence, as depicted in Figure 2.4. It is similar to the variant of Phase 1 with the one directional Path Edges. Those edges feature the same attributes as the original ones between Trip Nodes. For this variant of Phase 4 it is relevant to evaluate whether the original bidirectional edges between trips which are now connected with the one directional Trip-on-Route edges can be removed.

Sequencing as extra edge attributes

Many of the routes only consist of a single trip, as can be seen on 2.5c, where the Route Nodes for routes consisting of one trip overlap with the respective Trip Node. Therefore a dedicated type of edge could have insufficient data for weights to be learned for it.



(a) Representation, as planned in advance. The example assumes more trips than usually encountered on one neighborhood, and for clarity, omits most of the shipment nodes, to focus on more generalizing nodes.



(b) Representation of an actual neighborhood, taken from the data. In this example, there are only two routes that include more than one trip, both having two. This means there is only two sequence represented by a directed edge.

Figure 2.4: Phase 4, representing sequences of trips within a route. The example assumes a separate type of edge to showcase it.

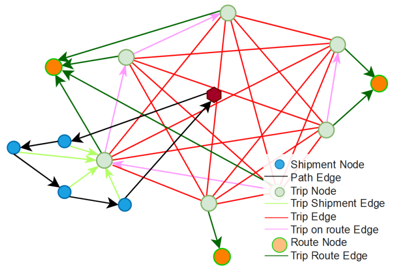
Instead, an alternative variant of the Phase is to add an extra boolean attribute to the Trip Edges from Phase 3. It uses the representation of bidirectional edges as two separate single-direction ones. When true, the meaning of the attribute would be "the destination trip is right after the origin one, and on the same route".

2.3.5 Phase 5: Route Nodes

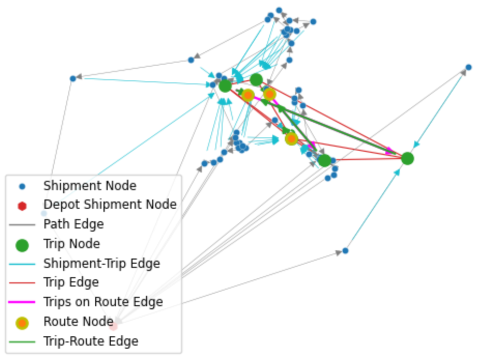
		Representation
Nodes	Route Nodes	Full routes (one or more trips) of vehicle throughout a day
Edges	Trip-Route Edges	Connecting Trip Nodes to the Route Node representing the route those trips are part of

Table 2.6: The elements of the graph included in Phase 5.

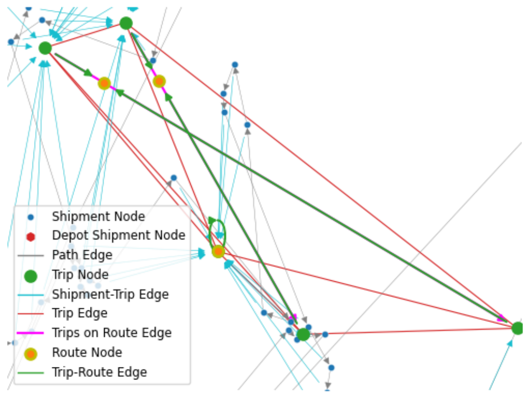
In Phase 5, the next step of generalisation is added, as the last node type within these designs: Route Nodes with respective edges, listed in Table 2.6. Route Nodes represent routes consisting of at least one trip, and can therefore be connected to an already existent Trip Node. They have multiple features, including coordinates (averaged over the trips in a route), distance, duration, costs and information regarding business violation such as lateness. As mentioned before, reflexive edges with dummy attributes are added from a Route Node to itself, to preserve own node information at message passing.



(a) Representation, as planned in advance. The example omits most Shipment Nodes, to focus on Trip and Route ones. The location of Route nodes with respect to Trip ones is creatively rendered, for clarity.



(b) Representation of an actual neighbourhood, taken from the data.



(c) Zoom on Trip and Route Nodes' representation from the real data from Subfigure 2.5b. 2 of the routes consist of 2 trips, and 1 only of one. For the latter, the route and trip nodes overlap, as they have the same coordinates.

Figure 2.5: Phase 5, Adding Route Nodes and edges from Trip Nodes to them

Edges are added between Trip and Route Nodes, with attributes like distance between their coordinates, duration and cost fractions of a trip on the route. If a route consists of a single trip, the distance between coordinates would be 0, as can be seen in Figure 2.5c, where one of the routes overlaps with its trip. In most cases a route consists of two trips, in which case its coordinates are exactly on the line between the two, seen in the same Figure.

Bidirectional Trip-Route edges

Just like with the edges between Shipment and Trip Nodes, the directions of the Trip-Route edge can be experimented with. A bidirectional edge would allow for information to flow not only from Trip to Route Nodes and back, but, in two steps each - from Shipment to Route nodes as well. In addition, it would allow information from all trips, including from other routes, to pass to each Route Node.

One directional edge from Trip to Route

Given that many of the routes have only a single trip, however, it is plausible for a bidirectional edge between single Trip and Route Nodes to cause back and forth message passing between the two. After aggregation, this can make the nodes similar and lose their individual properties (aside form their feature dimensionality). To prevent this, a one directional edge can be used. Similar to the reasoning with edges between Shipment and Trip Nodes, the direction of the single edge should be from the less general type (Trip) to the more general one (Route), as depicted in Figure 2.5a. It is expected that the Route Nodes then gather all the information from the graph as summary nodes.

2.3.6 Phase 6: Edges between Routes

		Representation
Nodes	-	-
Edges	Route Edges	Interconnectivity between Route Nodes

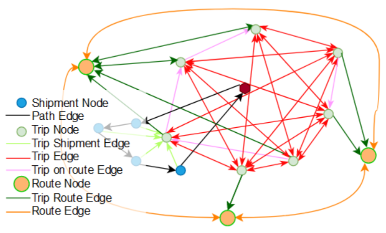
Table 2.7: The elements of the graph included in Phase 6.

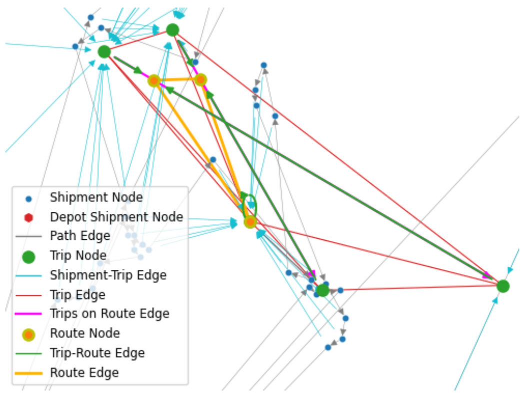
Similar to Phase 3, Phase 6 connects Route Nodes with bidirectional edges (per Table 2.7), as shown in Figure 2.6. No relevant features were deduced for the edges. The coordinates of the Route Nodes are dependent on trips, and it is possible for two trips on the same route to be in completely different directions with respect to the depot, making the one attribute of Trip Edges inapplicable to the Route ones. Route Edges allow for messages to be passed in one step from one Route Node to another. The reasoning for it is to present a way for each Route Node to share how they could contribute to improving another route's planning.

2.3.7 Phase 7: Empty Route Nodes

Finally, Empty Route nodes are added, requiring some alterations in the graph structure built so far graph - specifically in the elements mentioned in Table 2.8. As it was mentioned in Section 1.2, a route is 1-to-1 connected with the vehicle that is driving it. In some cases of real world application,a depot would have one or more idle vehicles not available, but not included in the planning. It can be used to relieve other routes, reducing penalty costs (like delays and overworking) but starting a new route has inherent costs (like the driver's wage and fuel consumption). Hence it is a matter of trade-off between the two costs to decide if using the extra route can improve the solution. Therefore, knowledge of available resources that can be used when necessary, is valuable for making informed decisions. Consequently, if it is available, the plan should consider inactive resources as empty routes. Even in a VRP with an available free vehicle, not all neighborhoods have access to the empty route, because of the neighborhood creation algorithm, not detailed here.

In this Graph Phase, Empty Route Nodes are added as Route Nodes, with most of their attributes (like duration and distance covered) being set to 0. Intuitively, their coordinates are set to match





(a) Representation, as planned in advance. The example omits most Shipment Nodes to focus on Trip and Route ones. The location of Route nodes with respect to Trip ones is creatively rendered, for clarity.

(b) Representation of an actual neighborhood, taken from the data, zoomed in on Trip and Route Nodes.

Figure 2.6: Phase 6, adding edges between Route Nodes.

the depot, as this is a consistent point between different neighborhoods of the VRP, where the vehicle is if it is not planned to move. The most intuitive method to add the Empty Route Nodes is to do it along with the edges connecting them to the other nodes from Phase 6. The nodes would therefore be bidirectionally accessed from all the other Route Nodes. An alternative is to add them along with empty Trip Nodes.

2.4 Graph type

The constructed graph has a number of properties that the eventual GNN will need to account for.

Primarily, the graph is **heterogeneous**. The entirety of the phase structure is based on handling additional node and edge types, and exploring how they will affect the performance of a consistent GNN. While in Graph Phases 1 the graph is still homogeneous, the same GNN will be used across all Graph Phases for the purpose of inter-comparison. Because of the variety types of nodes and edges throughout the Graph Phases, the GNN needs to support heterogeneous graph input.

Furthermore, the graph is **directed**. While some edges can be bidirectional, the graph by definition uses directed edges, to showcase dependencies (e.g., between levels of generality), and even most of the bidirectional edges, like the ones between Trip Nodes can make use of direction-specific information. Bidirectional edges are thus be represented as two directed ones in opposite directions.

The graph is also **static**, as opposed to dynamic. Dynamic graphs alter their states over time, and a GNN that handles them would need to accommodate for it during learning. This is not a requirement in the current case. Additionally, the developed graph is **not a hypergraph**, because each edge is between exactly two nodes. It is also **not a large graph**, as each example has a number of nodes and edges in the order of tens but hardly hundreds - hence it can easily be read in memory, and there is no need for partitioning due to space limitations. Even further, the graphs are small enough that they can be batched together for faster processing, as will be discussed in

		Representation
Nodes	Trip Nodes*	Trips vehicles make from the depot and back to make deliveries
	Route Nodes	Full routes (one or more trips) of vehicle throughout a day
Edges	Trip Edges*	Interconnectivity between Trip Nodes
	Trip-Route Edges*	Connecting Trip Nodes to the Route Node representing the route those trips are part of
	Route Edges	Interconnectivity between Route Nodes

Table 2.8: The elements of the graph altered in Phase 7. The ones marked with (*) are only altered in some variants.

Section 3.4.

Finally, the graph in principle makes use of **multiple edge attributes**, despite most features being assigned to nodes. Within LP, values are only known for shipments, trips and routes. However, one of the main reasons for experimenting with GNN on VRP is to explore the relations between those entities. To that end, some relationships require one or more parameters (such as coordinate-based distance approximation, real life distance, duration of travel, etc.).

2.5 Extracting graph representation

While running of Dassault Systèmes' LP, neighborhoods are dynamically created by selecting a few routes. From them, features are collected regarding the shipments' pickup and delivery, trips from the depot and back, and routes of the vehicles along with their features, as well as of the relation between them.

The implementation of the graph structure is LP, in Dassault Systèmes' internal language Quill. Once a neighborhood is created, the necessary properties for the graph are accessed from it, with the nodes and their features extracted from the properties. Each node is recorded with a unique identifier. Then, given two nodes, the respective edge between them is exported, marking its origin and destination with the nodes' identifiers, and its attributes are assigned from existent values in LP. Identifying nodes are always referred to as first and second within an edges, making all the edges one directional. If bidirectional edges are required, two one directional ones are taken, identical but for the order of identifiers.

All of this information is exported as a dictionary to a JSON file, using the respective formatting, where the primary keys are node and edge types. The value of each key is a dictionary where the keys represent feature names and their values are the feature values.

The resulting JSON files are then processed in Python [14], as detailed in the next chapter in Section 3.2. The visualisations of the graph data seen throughout this chapter are created with Networkx library [19].

Chapter 3

Methods: GNN Design and Implementation

When creating a GNN to solve a specific task, a number of design decisions need to be made about the network and its preliminaries: Defining the task, its in- and output, internal architecture and external application.

Section 1.3 discussed the general approach towards the problem. Already advised by Dassault Systèmes, it features two separate instances of a GNN, each of which creates the embedding of one graph. Both graphs are then compared. The process of applying this approach is discussed here: The task to be learned, described in Section 3.1; Data generation in Section 3.1.1; The tooling used in Section 3.2; Choice and usage of a GNN in Section 3.2.2; Design and implementation decisions in Section 3.3; And finally, batching and tuning in Sections 3.4 and 3.5 respectively.

3.1 Task

As described in Section 1.3, this project's goal is the creation of a model that estimates which neighborhood to select for optimization by the LNS-based VRP solver. To achieve it, a GNN-based model will be trained to select one of two neighborhoods depending on which one provides a larger virtual cost reduction after optimization. Within this research project, the chosen method is supervised learning which outputs the chances of its input being a member of one or another class. This is a binary classification and can thus be represented as stating whether the first neighborhood has greater optimization capacity than the second one in the pair, without loss of generality.

Then, in order to evaluate the model's performance during optimization, it is used at each iteration to compare all combinations of 2 out of n possible neighborhoods. Those comparisons are used to establish a ranking of best to worst expected optimization gain. The neighborhood with the best gain is selected to be processed by the solver. The ranking is relatively resistant to mistakes, as even if some of the comparisons are misclassified, the specific placements of each option in the ranking are not as important as ensuring that the overall more preferable neighborhoods are ranked higher.

Variation in the labels of the data points with respect to their features is the main reason for using this approach instead of a more direct one, like regression over the neighborhood embeddings. Since the iterations of the sub-solver yield different improvements depending on how early or late they are in the optimization process, collection of training data across the entire run is challenging. Additionally, data will be gathered on sub-problems from various different VRPs. Because of those differences, it is more difficult to estimate the improvement caused by each neighborhood's optimization as a stand-alone, than in comparison to other neighborhoods.

Therefore, the task to be learned can be summarised as the following: Given two neighborhoods, make a binary classification as to which one, when optimized, would result in a higher reduction in virtual cost of the VRP they are both part of. If they happen to be very similar and a good classification cannot be made then the potentially wrong result would not be of much significance.

3.1.1 Data Generation

Given the supervised training task at hand, a training data point needs to consist of a pair of neighborhoods and a label indicating which one of them has a better optimization capacity.

Within LP's implementation, neighborhoods are generated on demand, to be optimized by the sub-solver. Thus, the way to obtain both neighborhoods and their optimization capacity that is most true to real-world application is during the run of the VRP optimizer in LP. Each iteration of the sub-solver alters the state of the whole VRP and potentially changes how processing any other neighborhood would affect the virtual cost. In each one, a single neighborhood is generated sing non ML-based heuristic, and is optimized by the VRP solver. The neighborhood is saved and resulting improvement of virtual cost- measured.

Within the current research project, data is gathered from optimizing 7 different data sets, obtaining the neighborhoods for optimization and the measured improvement in virtual cost. Then a method developed specifically for Dassault Systèmes, ensures that pairs of neighborhoods are only made with neighborhoods in similar parts of the optimization both within and between problems.

In the resulting dataset, each data-point contains two neighborhoods, converted to graphs, labelled per Section2.3 as to which neighborhood is better for optimization, along with some additional information (such as importance weights). The dataset consist of about 10 000 pairs, combining neighborhoods from the 7 VRPs being solved.

3.2 Tooling

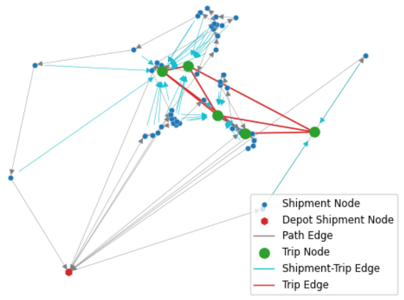
The current research uses the Pytorch-geometric library, built on top of the pytorch modules [24]. Pytorch-geometric is one of the state-of-the art libraries tailored for managing GNNs. It provides significant freedom in GNN design, as it features various types of graph convolution layers. Those are directly implemented from recent papers, and can be parametrized, mixed and matched with high inter-compatability [14]. Furthermore, Pytorch-geometric provides support for heterogeneous graph data and graph batching.

There is a variety of GNNs that can be leveraged, as discussed in the Literature Review in Section 1.4. In this research paper, the GNN used needs to abide by the requirement of input graph, stated in Section 2.4. The same holds for the method of representing the graphs themselves via Pytorch-geometric.

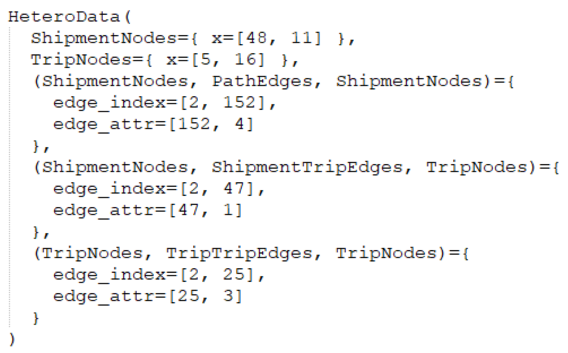
3.2.1 HeteroData

All the Graph Phases after Phase 1 consist of heterogeneous graphs, with multiple types of nodes and edges. To accommodate this, the two-layer dictionaries, described in Section 2.5 are represented in HeteroData format, available as part of the Pytorch-geometric module.

An object of HeteroData represents each type of node as keyed to the type's name, and valued with a matrix of all the nodes of that type's features (e.g., in Figure 3.1b it can be seen that there are 48 Shipment Nodes, with 11 features each). Every type of edge is defined with the types of nodes it connects and the name of the edge type, and has two sets of information: The indexes



(a) Visualization of graph from Phase 3



```
HeteroData(
  ShipmentNodes={ x=[48, 11] },
  TripNodes={ x=[5, 16] },
  (ShipmentNodes, PathEdges, ShipmentNodes)={
    edge_index=[2, 152],
    edge_attr=[152, 4]
  },
  (ShipmentNodes, ShipmentTripEdges, TripNodes)={
    edge_index=[2, 47],
    edge_attr=[47, 1]
  },
  (TripNodes, TripTripEdges, TripNodes)={
    edge_index=[2, 25],
    edge_attr=[25, 3]
  }
)
```

(b) Representation of the same graph from Phase 3 as HeteroData

Figure 3.1: Graph visualization to HeteroData representation

of the nodes (within their types) the edge goes from and to, and a matrix storing the features of each edges (e.g., in Figure 3.1b, there are 25 edges between the 5 Trip Modes, including reflexive ones, with each having 3 features). The HeteroData object does not feature undirected edges and represents bidirected ones as two one-directed (e.g. in Figure the 3.1b, there are 10 bidirectional edges between Trip Nodes, represented as 20 directional ones).

3.2.2 GNN layers' selection

Within the Pytorch-geometric library, there are various choices of implemented convolution layers for GNNs. The ones used in this research project needs to accommodate the characteristics of the graph design discussed in Section 2.4. The representation of the graph in Pytorch-geometric, described in Section 3.2.1 already incorporates only directed edges, and all of the module's GNN layers are appropriate for that.

Beyond that, Pytorch-geometric classifies the types of GNN layers based on the properties of the graphs they support. Table 3.1 outlines how the properties of the graph constructed in Section 2.3 are reflected in the requirements for the GNN layers.

Graph property	**GNN layer property**
Attributes on some edges	layer with edge attributes support
Directed edges	directed message passing
Graph is predefined (static)	later with static support
Heterogeneous	Homogeneous layers per type, collected in HeteroConv wrapper
Edges between different node types	bipartite layer

Table 3.1: Converting graph properties to requirements for the GNN layers.

Within the graph design, some of the edges are between two different types of nodes. Layers that allow for this are called bipartite. In Pytorch-geometric there are some layers that are bipartite, but are not heterogeneous, because they are limited to only two node types ([2]). Conversely, some options in Pytorch-geometric support an unlimited number of different types of either nodes or edges ([3], [30]), but are not bipartite.

The tool used to fulfil both these heterogeneous data requirements, is a HeteroConv wrapper for multiple layers. It collets a number of separate homogeneous bipartite (as needed) GNN layers, each of which can be customized towards specific edge type requirements. The message passed by those layers are then aggregated together per node. This solution is the one fitting best to the graph design in this paper.

In this research paper, the same type of GNN layer will be used for each edge type and aggregated by the HeteroConv wrapper. It needs to be **bipartite**, **static** and to support **edge attributes**.

Within the chosen library, there are a few convolutional layers that satisfy all those requirements implemented in Pytorch-geometric. Some of them are strictly dynamic ([36], [3]) but the GNN layers used need to support static processing (during embedding no edges or nodes are inferred), because graph was specifically crafted with message passing only over the predefined edges in mind. Others which interferes with the batching method, or are specialized for deep GNNs ([31], [15], [18], [40], [26]), or combinations of the two. In this research project, convolutional layers developed for a deep GNN are avoided, because the GNN being designed is planned to be relatively shallow, as argumented in Section 3.3, and hense the layers might need alteration. An interesting topic for further research is if deep GNN layers can be used with the graph designs discussed here.

Another layer fitting the requirements, which is already implemented within Pytorch-geometric is a General Convolutional Layer for GNN, originating from You, Ying and Leskovec's work [41] on designing a space for testing and parametrizing GNNs. The General GNN described there features basic message passing and is intended to function as a control option for performance of other GNNs on different tasks. It is therefore a perfect candidate for this research project, which aims to explore the influence of a GNN on VRP with neighbourhood selection in general, and not necessarily focus on selecting the best one for the problem. It is a matter of further research to investigate if any of the above-mentioned GNN layers, or others, would have a further beneficial influence to the problem. Deep GNN have some limitations and requirements mitigated by custom layers build, which might occur here as well.

3.3 Graph Neural Network Architectures

The next step after the GNN convolution layer is chosen, is to design the overall architecture of the network it is included within. Figure 3.2 outlines the general principle, with the note that some changes are needed between Graph Phases.

As discussed in Section 3.1, each data-point consists of two main parts - a pair of graphs and their label. The labels are only used to evaluate the model, ans well as by the loss calculation for the backwards pass of the network. In this case, backwards propagation is done via Binary Cross Entropy loss, as implemented in PyTorch [14].

In the architecture of the developed GNN, each graph is passed through its own set of GNN layers. These layers output two respective graph embeddings that need to be compared to each other. This section details the process.

3.3.1 GNN: Graph embeddings and gathering

A GNN layer in this setup is a Hetero Convolution layer, which aggregates a separate General Convolution Layer (GCL) for each edge type included in the respective phase. A GCL considers edge attributes by summing the result of a linear message passing layer and a regular linear layer which receives the edge attributes (or their embedding on later steps) as input. Given that the layers are exclusively edge dependent, reflexive edges need to be added to all the nodes (as already

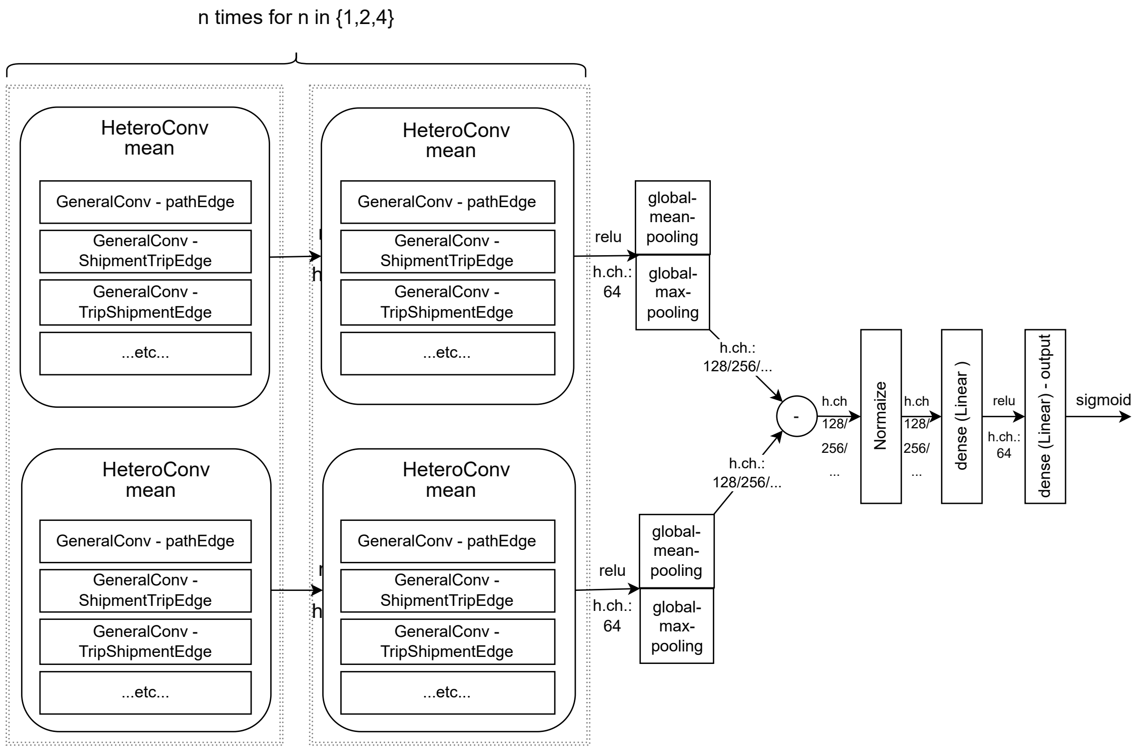


Figure 3.2: Architecture of the neural network

mentioned in Section 2.3), in order to ensure that the values of each node before the forward pass are considered in the update. Each of the GCL layers is activated before the global activation of the HeteroConv wrapper with a Leaky Relu function with 0.2 chance of using values below 0. As the GNN is constructed, each of the GCLs expects a known amount of edge attributes.

The HeteroConv wrapper aggregates the updated weights of each GCL layer. It was experimentally discovered that the best method to do this to take their mean. The alternatives were rejected as follows: Throughout experimentation it was observed that summation makes the scale of the layers' output values dependent on the number of types of nodes and is very susceptible to the varying scale of the graph element's features (some are in the fractions and others - in the thousands). Taking the maximum of the weights, in turn is observed as susceptible to differences in the scale of GCL output values. Both drastically increase the scale of output values. Conversely, a mean function of aggregation showed to allow for lack of scale divergences through multiple layers.

Each GNN layer passes information from one node to another, over a single edge. Thus, multiple layers are needed in order to have message passing over a few hops, as expressed in Section 2.3. Latter phases use three different types of edges with increasing levels of generalization, and at least two layers are needed in order to ensure information passes between all the levels. The exact even number of Hetero-GNN layers to be used will be inferred when tuning the model, as discussed in Section 3.5, at 2, 4 or 8. In this project, the influence of GNNs in general and their usage with different graph designs is being explored, and no focus is given to deep learning specific techniques.

Between each Hetero Convolution layer, a Relu activation is used. This is needed in order to perform activation of the aggregated GCLs in the HeteroConv wrappers after the internal leaky Relu of each GCL. The GCL-s in the first Hetero Convolution layer start with a number of in-channels, depending on the respective origin and destination nodes' features and the consequent ones have 64 hidden channels.

Passing the graphs trough a sequence of GCLs wrapped in HeteroConv layers yields the baseline graph embeddings, shaped as the input graphs. Because this research project aims to perform graph classification [12], the graph needs to be gathered next. Referring back to the Literature Review 1.4, that is the process of decoding from Section 1.4.3.

As the edges are used only for message passing, the notable information of the eventual embedding is stored in the nodes. This project combines two methods of graph gatherings: mean and max pooling. Each is done for every node type - first just form Shipment Nodes, then adding Trip and Route Nodes, as the Graph Phases progresses. All the mean and max poolings are concatenated, based on the matching hidden channel dimension, in accordance to the method in [23]. The so built matrix then has $2 * t * hc$ channels, where t is the number of different node types, and hc - the number of hidden channels.

3.3.2 Comparison

After the graph gathering the two graphs' representing matrices is of the same size of $2 * t * hc$. To compare them, a straightforward method is used, subtracting one from the other. The resulting single matrix is similar in shape and structure to the one that is achieved when a GNN is used to classify a single graph at a time, right after its gathering.

The last step, both here and in graph classification, is to render the obtained matrix into one binary output. This is done with two consecutive linear layers: The first reduces the number of channels back to the original hc and is activated with Relu. The second one outputs a single value, the result, activated with a sigmoid. It is the job of that final layer, to convert the all-positive values of the previous layer, to ones around 0, so the sigmoid can classify them.

Finally, a simple threshold comparison is made. If the output value is above a threshold of 0.5 without loss of generality, the first graph is predicted to be the better one to optimize, and if it is below it, the second one.

3.4 Batching

Section 3.1.1 mentions that the data labels are not necessarily perfect due to fluctuations in the state of the optimizer. Batching is utilized to avoid the model being specifically influenced by any specific data-point, and as an additional benefit, speeds up training. This paper uses a batch size of 512.

GNN layers only accept a graph as input, instead of a list of graphs. Pytorch-geometric provides a solution to this problem with a custom for the module data loader that turns a list of graphs (and matching labels) into one bigger graph, where the original graphs are disconnected sub-graphs. The node-based pooling layers, described in Section 3.3.1 are first scaled internally based on the information kept in the data loader, and then assign nodes to members of the batch. The model's output channels are now set to be equal in number to the members of the batch, their order matching the order of the graphs in it, which can then be compared with the respective labels within the batch.

A data loader made specifically for this project accommodates for the pairwise data by organizing lists of first- and second-of-the-pair graphs, for element-wise comparison between the pair of graphs. This is done by creating two lists of graphs where each of the pairs is split up between the lists, the index of the pair in the original list matching the indices of the individual graphs in the two new lists. Both lists are fed together into Pytorch-geometric's loader to create two large graphs for comparison.

3.4.1 Normalization

Normalization is needed due to the structure and meaning of the data. Some of the features are in order of thousands, which, once pooled together in the graph gather section, scale beyond what the eventual sigmoid activation function can handle. In turn, other features are factor or boolean values, set between 0 and l. The difference in scale does not allow for normalization between the features of each node and edge type. The intuitive alternative is then to normalize across the same features, within the same typed nodes, or edges within one graph. However, a single graph does not have enough elements of the same type to normalise over, and the scale of the graphs and their features can vary within a batch, rendering this solution impractical.

Instead, the scale of normalization will be learned as well as the graphs. Within Pytorch-geometric there are graph-normalization layers, but they do not support heterogeneous data. Instead, while a heterogeneous graph-based alternative can be a point of further research, this project opts for an already implemented option. Specifically, a standard normalization layer is added in the NN architectures right after the graphs are gathered and subtracted. It takes a known-sized matrix and normalizes it with learnable weights.

3.5 Tuning

A number of settings were tested when creating the model in order to decide on its parameters. A single number of epoch test is conducted as well as a grid search for learning rate and number of hidden channels in only one Graph Phase. The best performing settings are then used throughout all the Graph Phases, with the assumption that they will be acceptable, even if not ideal for them. It is outside the scope of this project to find the best possible settings, but just feasible ones (also demonstrated by just using GCLs within the GNN, instead of experimenting for the best type), explaining why no specific experiments are done to find the best parameters for each Graph Phase.

All parameter tuning is done on Graph Phase 2, where Shipment Nodes (with edges between them) and Trip Nodes (with reflexive edges and ones to Shipment Nodes) are included. This is the first phase where heterogeneous data is being handled, and hence allows for examination of the settings dependent on the presence of different types of nodes and edges. Conversely, of all the heterogeneous Graph Phases, this one has the fewest parameters, and would require the least amount of time to tune. These considerations make it the best option for a single-phase parameter tuning. All tests are done with a single seed for the random values on the algorithm, and are thus comparable.

First, a run with 800 epochs on this phase is executed. It is observed that the validation accuracy flattens after 80 epochs, and then stays consistent for 70 epochs, after which over-fitting begins. Therefore, using 90 epochs is considered suitable, as it provides limited accommodations for the increasing complexities of the following Graph Phases.

Then, with the set number of epoch, grid search is used to experiment with the learning rate (0.1, 0.01 and 0.001) and the number of hidden channels (32 or 64).The results are shown in Table 3.2.

The number of layers used are also a potential note of experimentation for future research. Because of some preliminary results, however, they are statically set to 4 for this research project, as that is a number with fairly consistent applicability over the Graph Phases. The expectation is that different number of GNN layers will be better applied for different phases. Having 4 allows for bidirectional message passing between all node types (if applicable) and does not repeat passes excessively in earlier Graph Phases.

Concluding those tests, model development proceeds with 4 Hetero Convolutional Layers, learning rate of 0.01, and 64 hidden channels.

LR	**HC**	**Val Acc.**
0.1	32	0.530596
0.01	32	0.519825
0.001	32	0.516224
0.1	64	0.527987
0.01	**64**	**0.534708**
0.001	64	0.534252

Table 3.2: Tuning in phase 2. Experimentation is done on Learning Rate (LR) and number of Hidden Channels (HC) and the decision is made to use the highlighted values.

3.6 Analysis

Section 3.1 discussed the task to train the models on. While the training and validation accuracy are the main guide when tuning the parameters detailed in Section 3.5, solving that task is not what can determine the eventual performance of the model within its real world usage. As Section 3.1 describes, the task for evaluation of the model is a ranking of a number n of neighborhoods.

This section will thus outline how a Graph Phase (from the ones described in Section 2.3) is processed, going through its model's Training phase, and then a Testing one where its performance is obtained.

3.6.1 Training phase

In principle, a model is trained and validated on train and validation data, respectively, and tested on another independent dataset, with the size ratio of Train-Validation-Test data being 80-10-10. When training the model in each Graph Phase in this project, the same separation is made, with the test set being a pre-separated chunk of the original dataset - thus also a set of paired neighborhoods represented as graphs. All of these actions will be considered here as Training phase, as later on a more elaborate Test phase will be applied.

Data

In advance, the data necessary for all the Graph Phases is gathered, in the manner described in Section 3.1.1. Data generation is done on the basis of 7 original datasets representing 7 different problems to be solved. As the optimizer is working on them using LNS, all the necessary information for all the Graph Phases is extracted from neighborhoods being optimized. Thus, this step is done only ones.

Then, for each Graph Phase, a sub-graph of each data point is selected, including only the nodes, edges and features for the respective phase. Those are then used to form the before-mentioned pairwise data.

Model

The GNN model is then implemented and trained, as per Section 3.3. It is done separately for each Graph Phase, with special attention that only the relevant edges have respective GCL-s, and that the number of attributes per edge are set. Some of the phases have options that add attributes to edges already added in previous graph phases.

The model is trained for 90 epochs, with settings decided on in Section 3.5. The training and validation metric can be used to gain a general idea for the training process, and in the next chapter, the accuracy on the (static) test set will be reported. It will further be referred to as a Train phase test accuracy value. However, given the inaccuracy of the labels, explained in Section

3.1.1, those values are not as representative to the performance of the model, as testing it in the real world scenario of recommending neighborhoods within LP's pipeline.

3.6.2 Test phase

Once the model is trained, the Test phase starts, where the model is used in a simulation of a real-world scenario. Three different datasets, representing three new problems are tested on, later on referred to as Problem 1, 2 and 3 respectively. The VRPs are solved with neighbourhood selection, where 10 different neighborhoods are candidates for optimization on each iteration. These are converted to a graph each (per the respective Graph Phase) and combinations of two neighborhoods are made. The model is then applied on each pair, in order ot compare them. The results of the comparisons is used to make a ranking on the neighborhood's optimization capacity. The neighborhood leading the ranking is selected as the one to be optimized by the VRP solver. Every next iteration repeats the process.

45 runs are executed at solving each problem, using 9 different machines, with 5 run per machine. To keep consistent testing time, each run is set to last for 60 minutes. Given the different machines can fluctuate in speed and efficiency, any comparison is made based on the number iterations made. The virtual cost at each iteration is averaged over the 45 runs, separately per datasets. Given that the datasets are very different, the results observed for each and not averaged over the three. In general the performance is compared at the end of the run, but for models of different efficiency a fixed iteration point is taken to compare them.

As seen in Figure 3.3, using a GNN or any other specialized method to select a neighborhood is bound to take longer time than not using one at all. Thus, within the hour, more iterations are made if no ML model is used than if one is applied for neighbourhood selection. While that is worth considering, it is not within the scope of this project to make the most optimal implementation of the GNN possible, thus the comparison is made on a consistent iteration. Further work in this direction would be beneficial in later research, to try and make the number of iterations with ML closer to the one without and discover the hard speed limitations.

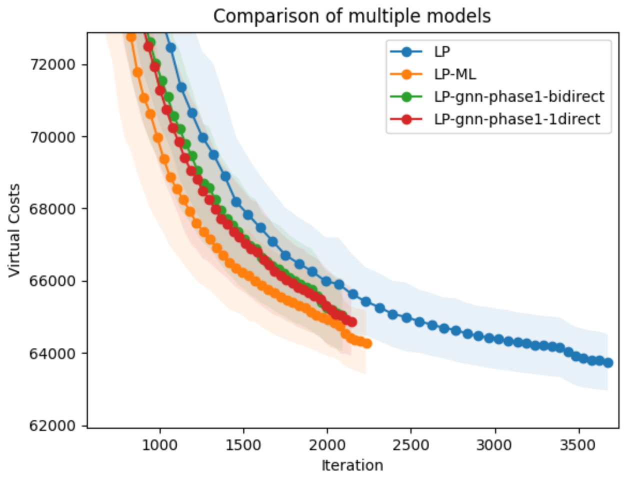


Figure 3.3: Example of the iterations of optimization, comparing the two options of Phase 1, Random Forest Classifier - the ML approach existing so far in LP, ML at all. The lower the virtual cost - the better the performance is. This example is of Problem 3.

Chapter 4

Results

The previous chapters list the different Graph Phases and their variation implementations, respective GNNs with comparisons trained on each of them, as well as how the model is tested on simulations of the real world VRP problem. This chapter discusses results obtained during training and testing, along with considerations of selecting which Graph Phases and variations to use.

The experimentation will be listed in two phases: The Training phase considers training the models on the task of selecting a more optimisable neighborhood out of a pair, listing their train, validation and test accuracy. The Test phase represents the use of the models in a real world-like scenario, evaluating their performance. Additionally, decisions of the Graph Phases are made in the Test phase, comparing them and two previously existing baseline methods within Dassault Systèmes' LP: Using random Forest classifier (RFC) for neighborhood selection and not using any neighborhood selection at all.

4.1 Training phase: Training, validation and testing

There are only 7 base Graph Phases listed in Section 2.3, but with their variations, there are 13 models trained, as depicted in Table 4.1. All models are trained on the same data using the same train-validation-test split, which is 80-10-10 respectively. Additionally, in order to compare the models, the initial weights of the networks are unified via the use of a constant seed for the randomisation. The implementation of the models is specifically structured per Graph Phase, as described in Section 3.3. All models are trained for 90 epochs with a learning rate of 0.01, 4 GNN layers with 64 hidden channels in both the GNN and the 2 Linear layers, and batches of 512 graphs.

In Table 4.1 are listed the accuracies at training, validation and testing during the Training phase of the model development. It only contains the Graph Phase combinations trained for testing in the Test phase. Table 4.1 shows that the highest accuracies of both train and test data (0.5420 and 0.5267 respectively) are in the simplest version of the graph - the one directional variant of Graph Phase 1. This model also has one of the highest validation accuracies (of 0.5502). This points towards simpler graphs being learned more successfully in the settings chosen. The idea is supported by the model being tuned on an early Graph Phase, and limited epochs and depth of the GNN being leveraged.

Conversely, the lowest training accuracy of 0.5056 is on the alternative variation of Phase 1. The lowest validation accuracy (of 0.5107), is on the variation of Phase 2 that introduces on-directional edges. Later validation results show that there is more to be gained from more complicated graphs, in comparison the the one in Phase 2. The highest validation accuracy of 0.5559 is in one of the variations of Graph Phase 4, the one with extra attributes added to the edges in Graph Phase 3.

Model	Train Acc.	Val Acc.	Test Acc.
Phase 1 one directional	0.5420	0.55024451	0.52675765
Phase 1 bi	0.5056	0.52727389	0.48427256
Phase 2 S-T, on Phase 1 bi	0.5118	0.51075810	0.49992966
Phase 2 bi, on Phase 1 bi	0.5251	0.53619515	0.50203204
Phase 3 on Phase 2 S-T	0.5151	0.54809266	0.48932749
Phase 4 extra attributes	0.5179	0.55837404	0.49642270
Phase 4 add edge type	0.5205	0.55598157	0.48082089
Phase 4 without Phase 3	0.5114	0.53906518	0.49679604
Phase 5 T-R	0.5209	0.54601401	0.50126492
Phase 5 bi	0.5194	0.54335451	0.48152315
Phase 6 on Phase 5 T-R	0.5228	0.55048650	0.48565641
Phase 6 without Phase 5	0.5196	0.54859997	0.50454437
Phase 7	0.5211	0.53228503	0.47551634

Table 4.1: Performance at Training Phase of the models in different Graph Phases. Abbreviations as follows: bi - bidirectional; S-T edges from Shipment to Trip Nodes; T-R - edges from Trip to Route edges.

It is of interest, that the lowest testing accuracy is on Phase 7, which has average train and validation accuracies in context. This points already towards the conditions of Phase 7 being highly situational, as will be detailed in Section 4.2.7.

On average, the training accuracy of all the modes is around 0.521, the validation - around 0.541 and testing - around 0.493. The accuracies on the test set of all the modes are around (including below) random values. A likely explanation for that and the difference from the validation and test accuracies is an unfavourable split of limited amount of data, which is consistent between models (because of a set random seed), given the fluctuation in the labels mentioned in Section 3.1.1. Furthermore the validation values were only observed as a method of preventing ovefitting at tuning (so only in Graph Phase 2) and do not influence training in any other way. Therefore, if the accuracy of the complete part of the data that is independent of the train set is to be taken, it would be the weighted average of the validation and test ones. That would put it closer to the train accuracy overall.

Some steps can be taken to overcome the limitation of inconclusive accuracy values in further research: Primarily obtaining more data, but also the usage of cross validation. Additionally, it was mentioned at tuning that the number of epochs for training (90) was selected based on Phase 2. In future research, longer training passes are recommended on the latter Graph Phases in order to ensure sufficient learning of more complicated graphs.

Even though the achieved accuracies seem low, all Graph Phases but Phase 3 showed consistent non-negative trend of both training and validation accuracies proportional to the epochs progression, implying occurrence of learning. The achieved accuracies of the models do not point towards a trustworthy model, but the next Section (Section 4.2) will show that using those models in real-world-like examples still causes significant differences in performance of the VRP optimizer. The combination of that Section and the limited results here show that the GNN already provides some direct comparison results with minimal training and there are signs of learning capabilities for a difficult objective.

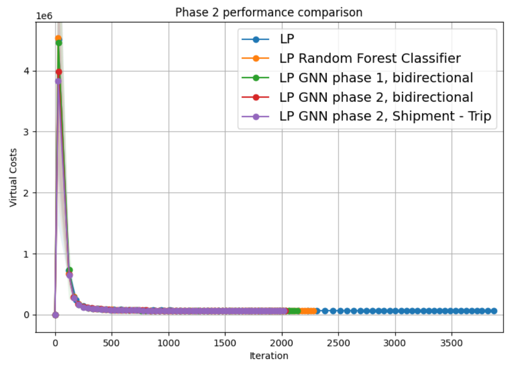
4.2 Testing phase and decision making

In the Testing Phase, the trained models are incorporated in the existing Last Mile LP optimizer. Three separate problems are examined, each optimized 45 for one hour.

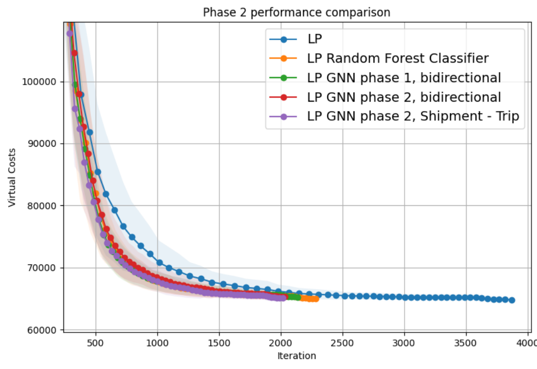
In its functionality, LP starts with no costs being estimated, setting the virtual cost to zero, followed in the first iteration where penalties for no routes are assigned, making the virtual cost extremely high, as can be seen in the start of the plot on Figure 4.1a. Then, once the optimizer with LNS starts working, the virtual cost drops sharply during the first few hundred iterations as routes get assigned and big improvements are made in the planninh. Only afterwards, as seen on Figure 4.1b, the method of neighbourhood selection affects the shape of the virtual cost decent.

Optimization with neighbourhood selection models makes considerably fewer iterations than optimization without any ML, as can be seen on Figure 4.1b. This is inherent from having multiple neighborhoods generated in the former case instead of just one in the latter. In that context, of interest is the achieved virtual cost at the end of the runs with neighbourhood selection, which can be observed between 1600 and 2400. Thus, in all the Graph Phases consequently discussed in this Section, the plots will be shown in that range.

Furthermore, all performance plots such as Figure 4.2 show that there is a sudden drop in virtual cost in the last few iterations, regardless of the neighbourhood selection model used. This happens due to a change in the objective made in LP in order to boost the final part of optimization. The drops happen at the same time points during the run, but their amplitude can be inconsistent between models. Comparisons will generally be made at the final iteration for the different models, but in some cases observations are to be made before the drop too.



(a) A full run of LP



(b) A full run of LP, with visible trajectories

Figure 4.1: Example full run of LP, on Problem 3, with Phase 2's performance.

This section shows the performance of each Graph Phase compared to its alternatives (its varaints as well as other Graph Phases an), and compares it to the baselines - LP without any ML based neighbourhood selection and RFC of the neighborhood's features, previously developed for LP. Each Graph Phase will begin with a table listing the elements included in this Graph Phase and its variants, similar to the ones in Section 2.3.

It is important to note the datasets are separate cases studies and the models have different performances between the three. Decisions will be made dependent on overall better performance, even if this performance is not the best in all individual problems.

4.2.1 Phase 1 - Basic Shipment Nodes and Path Edges

		Representation
Nodes	Shipment Nodes	Deliveries vehicles need to make and the depot
Edges	Path Edges	The travel of a vehicle from one location to the next (between Shipment Nodes)

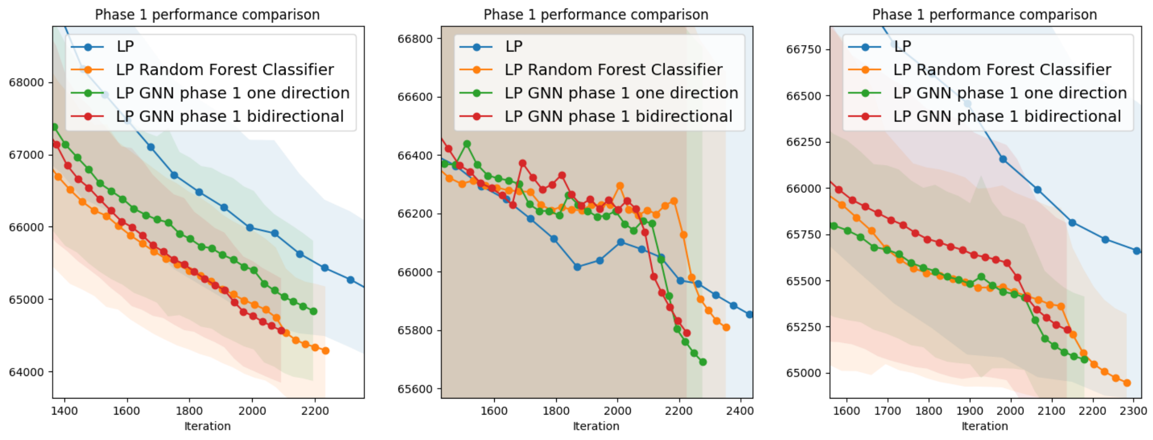
(a) The elements of the graph included in Phase 1.

Variant, per the plots	**Variant description**
Phase 1 one direction	Path Edges are only in the directions of travel of the vehicle
Phase 1 bi-directional	Reversed edges added to the ones in the previous variant.

(b) The variants of Phase 1.

Table 4.2: The elements of the graph and variants of Phase 1.

Per Section 2.3.1, Graph Phase 1 only includes the fundamental building blocks of the neighborhood as part of the graph representation (Shown in Table 4.2a). There are a lot of elements to add after this Graph Phase, and thus it is not expected to yield definitive results on its own, also shown in Figure 4.2.



(a) Problem 1 (b) Problem 2 (c) Problem 3

Figure 4.2: Phase 1 variations, compared to LP with Random Forest Classification and LP with no ML

The difference between the variations (one and bidirectional Path Edges) varies vastly between testing sets. In Problem 1, bidirectional Edges yield around 500 prior to the drop and 300 units at the end of virtual cost improvement from the one directional variant, while in Problem 3, 200 and 250 of deterioration respectively. In this comparison, Problem 2 fluctuates too much for any conclusions. Given that when using bidirectional Path Edges the increase in performance on Problem 1 is greater than the reduction on Problem 3 (as well as Problem 2), the bidirectional variant will be the one selected for Graph Phase 1. Additionally, on two out of three problems, that variant performs as well as the baseline of RFC and on the third - only 100 to 150 units worse.

To further argument this decision it is worth adding that when Section 2.3.1 described the bidirectional variant, it was considered as a method to overcome some of the limitations of the one directional one. Specifically, it enables propagation of information about scheduling issues from Shipment Nodes later in the respective trip, back to earlier ones. This choice has further implications of making the depot-node a summary of most of the other shipment nodes.

4.2.2 Phase 2 - Adding Trip Nodes

		Representation
Nodes	Trip Nodes	Trips vehicles make from the depot and back to make deliveries
Edges	Shipment-Trip Edges	Connecting Shipment Nodes to the Trip Node representing the trip those shipments are part of

(a) The elements of the graph included in Phase 2.

Variant, per the plots	**Variant description**
Phase 2 Shipment-Trip	Edges from Shipment Nodes to Trip ones
Phase 2 Bidirectional	Reverse Edges added to the previous variant.

(b) The variants of Phase 2.

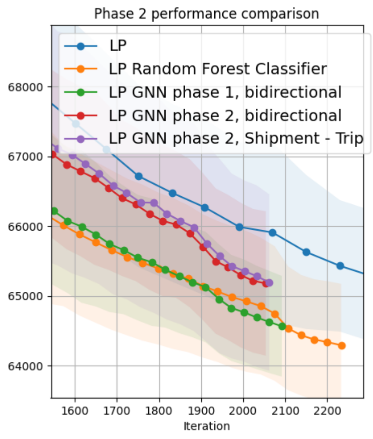
Table 4.3: The elements of the graph and variants of Phase 2.

In Phase 2, heterogeneous data is introduced to the graph with the elements listed in 4.3a. At least one edge is added to every Shipment Node (except the depot) which constitutes the largest increase in the graph size from all Graph Phases. This explains why an iteration on this Graph Phase takes slightly longer and thus fewer iterations are made in this Graph Phase than in Phase 1, as can be seen in Figure 4.3.

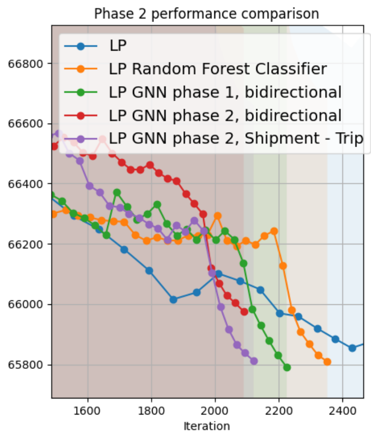
Figure 4.3 further shows that the bidirectional variation of Phase 2 (red) yield worse virtual cost reductions than the one achieved in Phase 1 - from just 50 units in Problem 3, to 200 in Problem 2 and 400 in Problem 1. A possible explanation is a discrepancy in usage of the two node types. As mentioned in Section 2.3.1, most of the properties of Shipment Nodes are requirements for the shipments: their time window, the distance between shipment, etc. In turn, Trip Nodes represent the plan being made and are thus flexible. In that context, it makes sense for the requirements, in general, to influence the plan, but not vice versa. Hence, Trip Nodes can be seen from a GNN perspective, as a summary node for all of its respective Shipment Nodes.

A further limitation of bidirectional edges is that they allow for Shipment Nodes within the same trip to pass messages between each other in two hops, via the Trip Node. This could cause the problem of over-smoothing, as when repeated enough times, each Shipment Node will receive information on all the other shipments multiple times, making it nearly identical to the summary node. It is interesting that this problem occurs so early, since, as seen later, it will become a frequent occurrence in the next Phases.

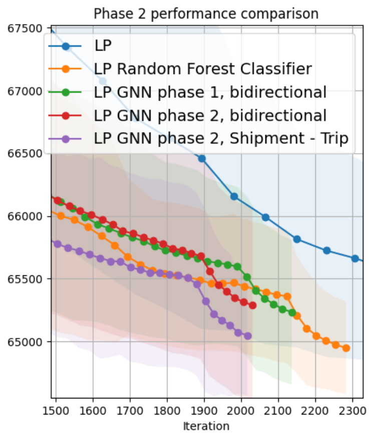
Alternatively, the one directional version of Phase 2 perform 200 units better than Phase 1 on Problem 3, on par but with fluctuation on Problem 2 and 400 units worse (similar to the bidrectional variant) on Problem 1. Given that this Graph Phase introduces Trip Nodes (a focal point for most of the following Phases), one of the variants needs to be used, as mentioned in Section 2.3.2. This will be the one directional one. Further research is needed to explore other ways to include Trip Nodes yielding more consistent results.



(a) Problem 1



(b) Problem 2



(c) Problem 3

Figure 4.3: Phase 2 variations, compared to LP with Random Forest Classification and LP with no ML

4.2.3 Phase 3 - Adding Trip Edges

		Representation
Nodes	-	-
Edges	Trip Edges	Interconnectivity between Trip Nodes

(a) The elements of the graph included in Phase 3.

Variant, per the plots	**Variant description**
Phase 3 Bidirectional	Bidirectional Edges between every two Trip Nodes

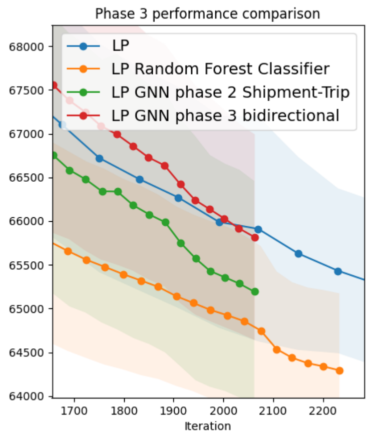
(b) The variants of Phase 3.

Table 4.4: The elements of the graph and variants of Phase 3.

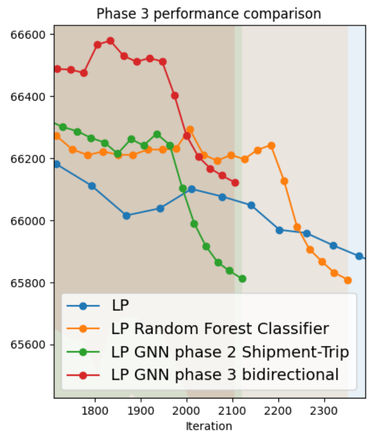
Figure 4.4 presents an interesting disparity: When designing the graph it was expected that message passing between trips (using the elements mentioned in 4.4a) would be essential for the graph representations, but it is instead a notable step backwards over all the datasets: 600 units in Problem 1 and 300 in Problems 2 and 3 in comparison to Phase 2.

A likely explanation for this is that when passing messages between all pairs of Trip Nodes in a neighborhood, the model makes all Trip Nodes in the same graph similar. This over-smooths the graph and not only loses its ability to differentiate between those nodes, but also the capability to learn specific features required to differentiate the two graphs being compared. Given that in this Graph Phase each Trip becomes connected to each other with direct edges, each Trip Node get information of all the others within one hop over a layer. On average, a neighborhood has 5-6 trips, and after the 4 GNN layers used, the information on different trips becomes very alike.

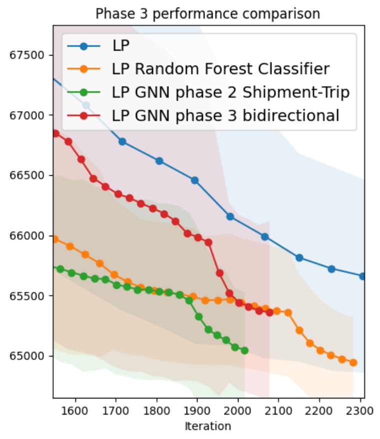
Trip-representing messages are already passed around at the depot Shipment Node, because of



(a) Problem 1



(b) Problem 2



(c) Problem 3

Figure 4.4: Phase 3 variations, compared to LP with Random Forest Classification and LP with no ML

bidirectional Path Edges chosen in Graph Phase 1. Thus, the relationship between trips is likely already extracted and connecting the Trip Nodes does not add enough learning power beyond that to compensate the heavy introduction of over-smoothing.

Furthermore, problems can be observed in the training process of Phase 3's model, partially shown in Table 4.5. While the training accuracy keeps rising, the validation one peaks at 55 to 65 epochs, and then decreases, implying over-fitting, whereas none of the other models show such clear signs.

Epoch	**Train Acc.**	**Val Acc.**
01	0.5026	0.5318795443
...	...	...
40	0.5086	0.5657643676
45	0.5103	0.5521492362
50	0.5113	0.5428577662
55	0.5109	0.5719668865
60	0.5124	0.5627813339
65	0.5123	0.5721082687
70	0.5127	0.5555395484
75	0.5134	0.5555008054
...	...	...
90	0.5151	0.5480926633

Table 4.5: Training of Graph Phase 3.

These two related problems can be tackled in a few different ways: "early stop" at training could help, but the issues would benefit more from structural improvement. Some options include using fewer layers for Trip Edges, or a variant of GNN layers that includes messages passed form other trips with lower weights than the own node information. These are points for further development, as within this project, it will be considered that Phase 3 needs not be added to the design.

4.2.4 Phase 4 - Adding sequence to Trips

		Representation
Nodes	-	-
Edges	Trip Edges	Interconnectivity between Trip Nodes
	Trip-on-Route Edges	Connecting sequential Trip Nodes that are on the same route

(a) The elements of the graph included or altered in Phase 4.

Variant, per the plots	**Variant description**
Phase 4 Attr. on Trip Edg.	Attribute to Trip Edges to mark sequence of trips on the same route
Phase 4 Trip on R. Edg.	Edges between Trip Nodes that are on the same route
Phase 4 no Trip Edg.	Edges for sequential same route, but no Trip Edges

(b) The variants of Phase 4.

Table 4.6: The elements of the graph and variants of Phase 4.

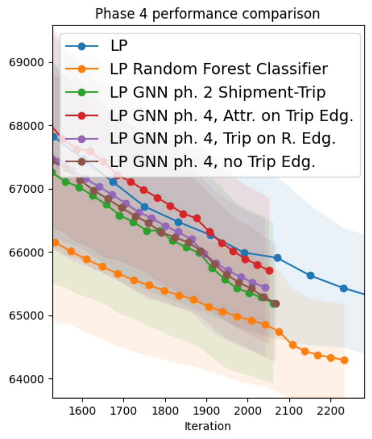
Omitting Phase 3 places some limitation on Phase 4, since they are interdependent. The variants planned in Section 2.3.4, both building up on the edges in Phase ,3 will still be tested. Added for testing is a variant that avoids Phase 3 completely by including the additional types of edges representing trips on the same route, but not the Trip Edges from Phase 3, as shown in Table 4.6b. All of the variants perform better than Phase 3, and hence the comparison of interest is to the previous Graph Phase used - Phase 2.

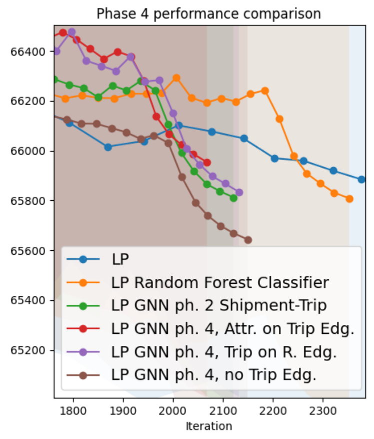
In the variants building on Phase 3, shown in Figure 4.5 as red and purple, the changes in Trip Edges mitigate to an extent the issue of over-smoothing that occurs when Trip Edges are added in the previous Graph Phase. Beyond the variant-specific changes, all Trip Edges are altered in Phase 4, to mark the duration between trips, even on different routes, in positive or negative numbers, depending on their order. Adding just extra attributes to Trip Edges (red in Figure 4.5) shows 600 units worse performance than Phase 2 in Problem 1, 100 in Problem 2 and 250 in Problem 3. Including Trip-on-Route edges instead still yields worse performance than Phase 2, but less so: 200 units in Problem 1, just 20 in Problem 2 and 100 on Problem 3. These results facilitate the decision to discard Phase 3 complexly.

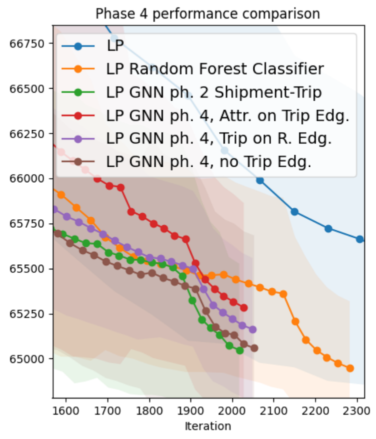
In turn, incorporated sequence of trips without the edges from Phase 3 (brown in 4.5) improves performance in comparison to Phase 2 by 200 points on Problem 2, and 100 on Problem 3 before the drop and is similar at the end of Problem 3 and throughout Problem 1. This shows that the variant indeed avoids introducing heavy over-smoothing.

That is ensured by the Trip-on-Route Edges being one directional, following the sequence of the Trips. Sequential direction of the edges is more applicable here than for the Path Edges in Phase 1, because most routes have only 1 to 3 trips, which is fewer than the layers handling the respective path edges. Additionally, the Trip-on-Route edges do not make a loop like the Path Edges do with a single connection point. Thus, while message passing could blur the last Trip Node on a route (making it a summary of all the trips on that route), earlier ones still have mostly their own information. Because of the limited number of trips, however, bidirectional sequence edges would over-smooth just like the edges introduced in Phase 3.

This Graph Phase and the previous ones show that bidirectional edges are good in long sequences







(a) Problem 1 (b) Problem 2 (c) Problem 3

Figure 4.5: Phase 4 variations, compared to LP with Random Forest Classification and LP with no ML

between high numbers of nodes, but cause over-smoothing on low numbers. This can be prevented by single direction edges.

4.2.5 Phase 5 - Adding active Route Nodes

		Representation
Nodes	Route Nodes	Full routes (one or more trips) of vehicle throughout a day
Edges	Trip-Route Edges	Connecting Trip Nodes to the Route Node representing the route those trips are part of

(a) The elements of the graph included in Phase 5.

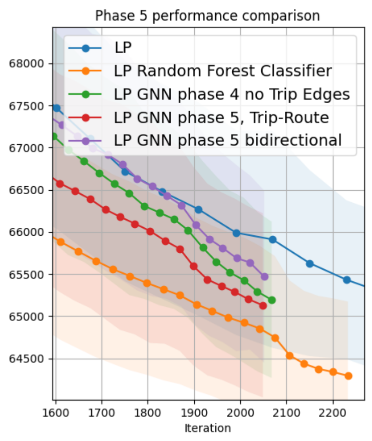
Variant, per the plots	**Variant description**
Phase 5 Trip-Route	Edges from Trip Nodes to Route ones
Phase 5 Bidirectional	Reverse Edges added to the previous variant.

(b) The variants of Phase 5.

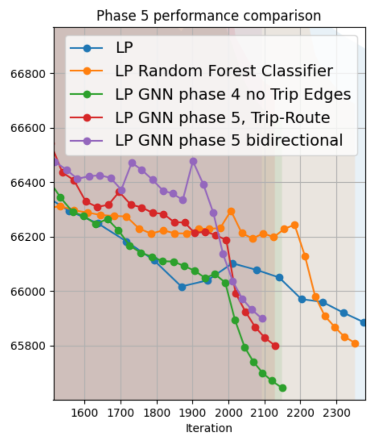
Table 4.7: The elements of the graph and variants of Phase 5.

Phase 5 introduces Route Nodes, which represent active routes, and Edges to them from Trip Nodes, as listed in Table 4.7a. Two variants are tested, as shown in Table 4.7b with one and bidirectional Edges from Trip to Route Nodes.

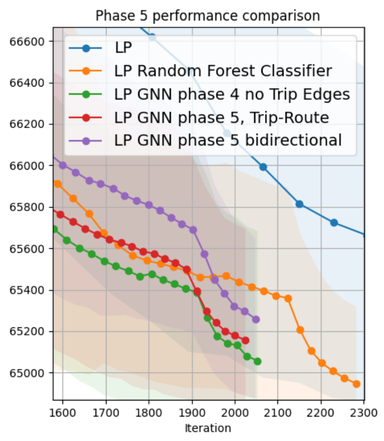
In Figure 4.6, it can be seen that the bidirectional case (purple) does not perform well: In comparison to the best variant of Phase 4, the performance is 300 units worse in Problems 1 and 2 and 20 in Problem 3. A likely explanation is once again the occurrence of over-smoothing, further supported by many of the routes having a single trip. In those cases, the Route and its respective Trip Node pass their information back and forth repeatedly. Despite outside information being passed to Trip



(a) Problem 1



(b) Problem 2



(c) Problem 3

Figure 4.6: Phase 5 variations, compared to LP with Random Forest Classification and LP with no ML

Nodes, there are still multiple repetitions of similar values being passed between Trip and Route nodes for each layer used, resulting in over-smoothing between the two types of nodes.

The other variant of the Graph Phase uses only directed edges from Trip to Route Nodes and is marked with red in Figure 4.6. It performs better than the first, reaching 50 units improvement than Phase 4 in Problem 1, and only 250 and 100 units worse in Problems 2 and 3 respectively. This points towards limited mitigation of the over-smoothing issue when using the Route Nodes as summaries of Trip ones.

A possible reason for the worse performance of both Phase 5 variants than the Phase 4 is Route Node feature selection of Route Nodes. They are the fewest in number of the three node types, but their pooling in the GNN has as big weight to the overall result as Shipment and Trip Nodes. Consequently, each of their features, including non-informative ones, if any, have greater influence as well. Because of that, it is interesting for future research to examine how informative each feature is, as well as different methods to connect Route Nodes to the graph, like having them receive information from Shipment Nodes directly.

The examined methods for inclusion of Route Nodes does not seem to be beneficial, but information in them is still important in the work of LP. The next Graph Phase will test if combining it with further extensions increases the benefit of Route Nodes when compared to skipping it altogether.

4.2.6 Phase 6 - Adding Route Edges

This Graph Phase introduces edges between the Route Nodes. As evident in Table 4.8b, a variant is tested, where that is added ontop of Phase 5, and one where Route Nodes are included disconnected from the rest of the graph, accounting for Phase 5 not being beneficial.

Figure 4.7 shows that having the interconnected Route Nodes connected to the rest of the graph is more beneficial than them being a disconnected clique [8]. The latter (purple) performs 400 units worse than Phase 4, in Problems 1 and 2 and 200 in Problem 3. Conversely, the former variant (red) performs 40 units better than Phase 4 on Problem 1 and is equivalent at the end of the runs in the other two problems. Interestingly, in Problem 3, before the drop of the run this variant has 200 units better performance than the other one, at only 100 worse than Phase 4.

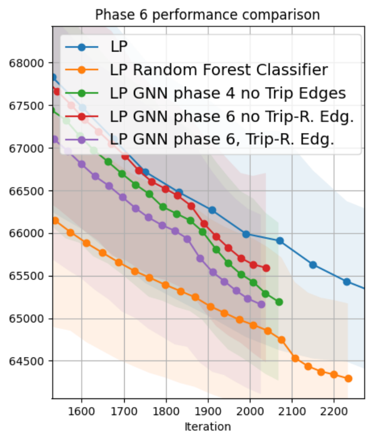
		Representation
Nodes	-	-
Edges	Route Edges	Interconnectivity between Route Nodes

(a) The elements of the graph included in Phase 6.

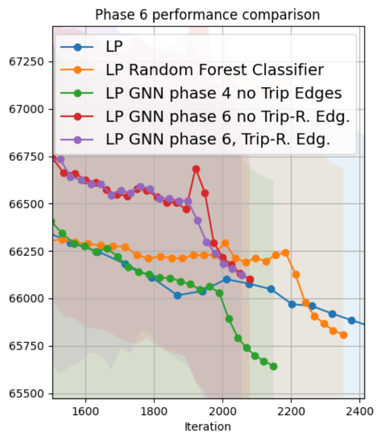
Variant, per the plots	**Variant description**
Phase 6 no Trip-R. Edg.	Interconnected route Nodes, disconnected from the rest of the graph
Phase 6 Trip-R. Edg.	Interconnected Route Nodes, with Edges between Trip and Route Nodes

(b) The variants of Phase 6.

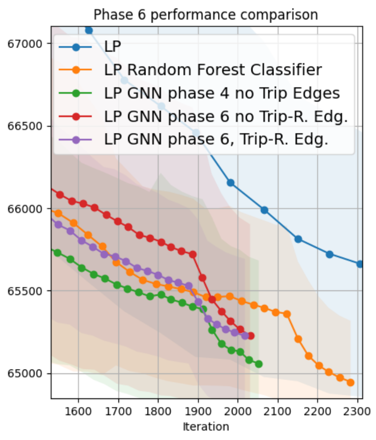
Table 4.8: The elements of the graph and variants of Phase 6.



(a) Problem 1



(b) Problem 2



(c) Problem 3

Figure 4.7: Phase 6 variations, compared to LP with Random Forest Classification and LP with no ML

The worsening in performance is once again explainable by over-smoothing: If Route Nodes are interconnected, they exchange information between each other and blend together similar to the Trip Nodes in Phase 3. Without connections of each of them to their respective trips, there is no way to retain information to identify individual Route Nodes and their differences from one another, which exacerbates the problem further. After one pass (out of four) each Route Node is indistinguishable, because no external information can enter the clique. The effect of the issue is smaller here than in Phase 3, however, because there is more information in the graph, that is not subject to over-smoothing.

Section 4.2.3 discusses ideas to avoid that over-smoothing, and this can be applied here too. A further approach to mitigating over-smoothing, coming up at this stage of the node hierarchy, is using one directional edges from the Route Nodes to a separate summary node. In this case, at graph gathering, both the information of separate Route Nodes and of a node summarizing them will be processed. However, in its current form this Graph Phase alone is not beneficial.

4.2.7 Phase 7 - Adding Empty Route Nodes

		Representation
Nodes	Trip Nodes	Trips vehicles make from the depot and back to make deliveries
	Route Nodes	Full routes (one or more trips) of vehicle throughout a day
Edges	Trip Edges*	Interconnectivity between Trip Nodes
	Trip-Route Edges	Connecting Trip Nodes to the Route Node representing the route those trips are part of
	Route Edges	Interconnectivity between Route Nodes

(a) The elements of the graph altered in Phase 7. The ones marked with (*) are planned as potential alteration, but are not concerned in the eventually tested variant.

Variant, per the plots	**Variant description**
Phase 7	Empty Route Nodes, with Edges from respective Empty Trip Nodes

(b) The variants of Phase 7.

Table 4.9: The elements of the graph and variants of Phase 7.

While neither Phase 3, Phase 5 nor Phase 6 showed to be value-enhancing additions to the graph, at least one of them needs to be included in order to examine the influence of adding Empty Route Nodes. They need to be connected to the rest of the graph in order to influence it. Within this research project's design philosophy, this can be done either by interconnecting them with Route Nodes or, via supplementary Empty Trip Node, by interconnecting them with Trip Nodes.

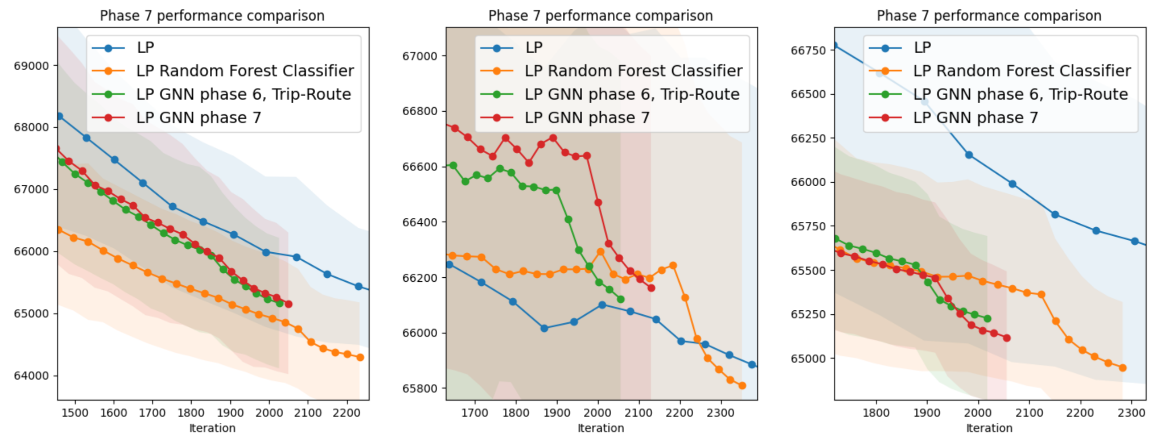
Given that Phase 6 (interconnecting Route Nodes) yielded better performance than Phase 3 (interconnecting Trip Nodes), only the variant of Phase 7 that builds on top of Phase 6 will be used, as mentioned in Table 4.9b. For each empty Route Node, an empty Trip Node is added, which aims to reduce smoothing over the Empty Route Nodes - if it is interconnected with the other Route Nodes and there is no independent source of empty information, its effect would be significantly reduced.

Figure 4.8 shows quite inconsistent results: Equivalent performance to Phase 6 in Problem 1, 50 units worse in Problem 2 and 100 better in Problem 3. This points towards the influence of those nodes being highly dependent on the specific use case and whether there are or enough instances with Empty Route Nodes.

Contrasting other Graph Phases, where a decision could be made as to the benefit of their addition and suggestions as to how and where to improve them could be made, this Graph Phase requires more test cases in order to determine the exact effect of adding the elements. Here, it does not appear beneficial, but, given the value in the LP, it might be worth considering alternative designs to add information about these nodes.

4.2.8 Global

After examining all the planned Graph Phases, it can be concluded that out of them, Phase 4 performs the best. More specifically, the best graph structure was determined to be the variation of Phase 4 with separate type of Edges that connect Trip Nodes on the same route, following the direction of their sequence. There Phase 3 is skipped, one directional edges from Shipment to Trip



(a) Problem 1 (b) Problem 2 (c) Problem 3

Figure 4.8: Phase 7 variations, compared to LP with Random Forest Classification and LP with no ML

Nodes are used for Phase 2 and bidirectional ones for Phase 1. Figure 4.5 (brown) shows that it provides 180 units of virtual cost improvement on Problem 2, equivalently on Problem 3 and 1000 units worse on Problem 1, in comparison to the already established RFC.

This conclusion is the result of decisions on Graph Phases described in this Section, some of which were made with lower certainty than others. Phase 1 is the most fundamental one. An argument can be made for testing the influence of the alternative variant on the following Graph Phases. Furthermore, it is interesting to note that for Phase 5, Phase 6 and Phase 7 a fair attempt is made to include a useful combination of their elements, but they all proved to not improve over Phase 4 in this setup. However, Route Nodes are important part of LP structure and warrant further examination.

4.3 Design lessons

It is observed throughout all the Phases that, when adding new elements to the graph design, constant attention needs to be paid to avoiding over-smoothing. The GNN needs to make a graph embedding of a neighborhood, such that it can be differentiated from another, but one neighborhood consists of a number of different routes, trips and deliveries, which need to be embedded with a degree of separation that specifically encodes the differences between them.

While this is done in specific manners through the Graph Phases, some principles emerge.

1. **Within nodes of the same type, fully connecting all pairs of nodes quickly leads to over-smoothing.** That is a reasonable conclusion, given that in a fully connected set of nodes, message passing over even one layer of GNN will pass the information of all the nodes to each one. Thus, especially if a node's own data is not treated in dedicated manner, they will all blend together.

   Instead, within the node type, another property needs to be found in order to establish relation between some, but not all, nodes (e.g., trips on the same route). It is preferable for this property to be transitive in some way, like the sequences of Shipment Nodes and Trips on the same Route, because message passing in a GNN is transitive in its own right. It is not a good idea to cluster nodes in interconnected sub-sets (as then the same problem occurs on a smaller scale).

2. **When it comes to edges of the same type with transitive relations, the directionality of edges matters.**

   One directional edges, following the direction of transitive relations, are good for short chains of relation. Specifically, it is good for chains shorter or equal to the number of layers. That allows for information to be accumulated over the tail end of the chain.

   Conversely, bidirectional edges are better applied in longer chains. This allows complex embeddings to be made across the chain while not being affected by the problem of over-smoothing too much. Furthermore, this keeps message passing consistent in different parts of the chain (contrary to e.g., the first node of the chain only receiving messages from itself).

   Furthermore, if the transitive relation is circular, and specifically if there are points where different circles intersect, those points need to be handled with special care. The reason is that the information from one chain can pass to another. The specifics of this are a matter of further research.

3. **When possible, edges between different node types need to be in the direction from less to more general.** More general nodes will then act as a summary of more specific ones. This works better when the more specific nodes are considerably more numerous than the more general ones, making the summary node very different form each separate one of the more specific nodes.

   In this case, bidirectional edges are not recommended, because then each of the more specific nodes becomes influenced by the summary of all the others, achieving the same effect as interconnecting nodes in two layers instead of one.

4. **The relationship between Shipments and Trip Nodes is not similar enough to the one between Trip and Route Nodes to treat the two as the same without further consideration.**

5. Because of the way the graph is gathered on a node level, **it is important to ensure node attributes are informative**, especially on the node types that are few in number. While not attempted here, a method to test if a feature on a node is needed, such as limiting testing condition to just the respective node type and observe performance with and without it.

Chapter 5

Conclusion and future work

So far, a method for using GNN for selection of one out of multiple potential neighborhoods of a VRP, has been developed and tested. Each neighborhood is a sub-problem of VRP and the selected one is optimized by a sub-solver. The goal is to use the selection method across multiple iterations of the sub-solver in order to solve VRP in a manner similar to the LNS approach. The problem was formally defined and a design for the graph representation of a neighborhood was detailed. Then training and testing tasks for the use of the GNN were outlined, followed by the design and creation of the GNN and its method of comparing neighborhoods. Finally, the results were analysed and pointers for approaching the problem with this method were produced.

5.1 Research Question: Answers

Within this project, a number of research questions were asked. Their answers achieved within the research are summarised below.

First, it was questioned **what makes a good graph representation of a neighborhood from VRP.** A good representation is an input for training and using a GNN to compare different neighborhoods in terms of the cost reduction optimization would provide, where this reduction is maximized. Experimentally, a variant of Phase 4 from the iterative graph representation framework developed proved to perform the best. Shown on Figure 5.1, the phase represents a graph featuring only Shipment and Trip Nodes, with bidirectional Path Edges, one-directional Shipment-Trip Edges and one directional Edges depicting the sequence of Trip Nodes on the same route, but does not include Trip Edges between all trips.

Second, the question as to **how to compare graph representations of neighborhoods** was posed. This was answered using the architecture of NN, described in Section 3.3. A pair of neighborhoods to be compared is processed simultaneously - each one embedded by a separate GNN part of the network. Then the graphs are gathered (similar to graph classification tasks) into matrices of the same shape, which allows for a straightforward subtraction-based comparison, followed by a normalization and linear layers. In this supervised learning model, the two neighborhoods are marked as first and second, and a binary label stating, without loss of generality, which one of the two has the highest optimization capacity. This allows for learning neighborhoods' qualities in the context of their comparison in terms of optimization capacity.

Next, it was asked if **a GNN provides additional value in terms of virtual cost minimization when used in real-world scenario of optimization of VRP**. The answer of these questions is highly dependent on the specification of the GNN, its incorporation in the comparison NN and the graph representation it is applied to. Figure 4.5 shows that the best variant of

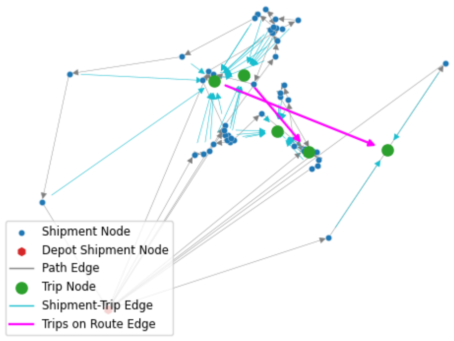


Figure 5.1: Visualization of a neighborhood in Phase 4, which performed the best

Phase 4 does indeed provide additional value in comparison to no ML methods for neighbourhood selection in all cases of VRP experimented on: 180 units of Virtual Cost improvement on Problem 2, equivalently on Problem 3 and 1000 units worse on Problem 1, in comparison to the already established RFC, and consistently equal to the performance of LP without neighbourhood selection as evident in Figure 4.5. Using three vastly different case studies makes performance fluctuation an expected occurrence but is good grounds for generalization, pointing towards this method being comparable in performance to RFC. As additional value of the method, this study outlines a big space for further improvements of the graph representation, GNN and comparison network design, which can yield further improvements. Thus, it can be concluded that using GNNs for comparison, even in the current stage, does provide additional value for cost minimization of VRP in real-world scenario.

Another question asked was **if a transformer would perform better with the same terms**. Within this context, a transformer is discussed as a specific type of GNN, that considers the input graph as fully connected. The research project, examines any GNN, and it is a point of consideration for potential further research as to which type and architecture of the GNN would work the best. Therefore, it is outside the research's scope to experiment with different GNN implementations, including transformer, to provide an extensive answer to this research question and the one on preferred GNN type. Without testing it specifically, however, it can be noted that treating the graph designed here as fully connected as to apply a transformer, is inadvisable due to reoccurring problems with over-smoothing as soon as nodes are interconnected.

5.2 Limitations and future research

This research explores the novel approach of selecting neighborhoods to be optimized by a VRP solver using a GNN. While some positive results were found, as shown in Chapter 4 and in answering the research questions in Section 5.1, its scope was limited due to the need for a combination of various techniques with extensive variations.

5.2.1 GNN selection and architecture

The biggest limiting factor is the selection of the GNN. Within this research, the overall influence of a GNN was explored, which can vary vastly depending on the GNNs architecture, layer implementation and its combination with other NN elements. This research project uses a general graph convolution method that implements only the most basic properties of a GNN: Message passing over the edges and aggregation of messages in nodes. There are many structural variations of a GNN that have the potential to improve the performance of the currently described model further (e.g., GAT [36] and transformer GNN [12]).

Additionally, this research project uses heterogeneous graphs which require aggregation of separate GNN layers per edge type. It assumes a uniform architecture over the different graphs, but that does not need to be the case. A GNN for heterogeneous data can make use of different layer combinations for each edge type in order to customize embedding depending on the types' specific properties. Further experimentation is needed to explore which type of edge requires what architecture, beyond the guidelines inferred in Section 4.3.

One approach to overcoming both those limitations is freely supported by the tool used in this project: Using different GNN convolution layers for different edge types. Pytorch-geometric [14] allows for a vast choice of layer and architecture combinations that can be tailored to data description in future works, where improvements in performance beyond the one achieved here are expected. To best explore the variations, a GNN comparison framework, such as the one in [41], can be used, as well as real-world simulation, like the one in the test phase discussed here.

5.2.2 Graph representations

While this project introduced an elaborate iterative graph representation framework, the options for Graph Phases are still limited. In Section 4.3, some conclusions were made as to how to design a good graph representation of a VRP sub-problem (here referred to as a neighborhood), based on the results of tested Graph Phases. Using them, new and improved graph representation (or multiple variations) can be made and tested to ensure all well informative data and graph structure is used. It is worth keeping in mind that the graph representation is highly dependent on the specific type of VRP as well as the available information of its elements.

5.2.3 Preventing over-smoothing

Over-smoothing is the most persistent problem encountered when using the GNN in this project. Many of the tested Graph Phases proved to be very prone to that issue. Within the current GNN implementation, the only way each node's own data is incorporated is by passing it as a message over a reflexive edge, which is treated the same as other incoming messages at aggregation. In this research project, the most vulnerable to the problem Graph Phases are removed and Section 4.3 lists some observations for future graph designs that could be less susceptible to it. Further investigation of its occurrence and solutions are needed.

Some methods to be examined include limiting the amount of layers for edge types especially prone to over-smoothing, possibly with use of specialized graph structures. Furthermore, a main starting point for this investigation is altering the GNN convolution layers to a layer type that differentiates between its own and incoming messages for each node. This can be done within the current implementation by weighting the messages or using a structurally different type of GNN (e.g., [3]). There are a number of papers ([26], [15], [40]) that explore novel approaches avoiding this problem e.g., to deep GNNs.

5.3 Conclusion

This project explores the use of GNN for neighborhood selection in VRP. It suggested using it with a VRP solver which improves the solution iteratively, optimizing one sub-problem (a neighborhood) of the VRP per iteration. The goal of neighborhood selection is for the VRP solver to make the largest possible steps per iteration that contribute towards the ideal solution. This is achieved by selecting the neighborhood that, if optimized, will yield the largest reduction in total problem cost. Within this approach, candidate neighborhoods are created and the GNN is used as a tool to analyse and compare them in terms of their optimization capacity.

The results of the best performing phase (depicted on Figure 5.1) proved to be quite promising. Numerous insights are gained for further implementation, where improvements to the graph representations and GNN architecture are concerned.

This project was a successful proof of concept showing that GNNs can benefit the performance of a VRP sub-solver via neighborhood selection. It can serve as a starting point to warrant further research on the topic.

5.3.1 Generalization and relation to AI

This research project is firmly set within Machine Learning, an inseparable part of Artificial Intelligence. It introduces a novel way to incorporate supervised learning within the domain of VRP, usually a difficult task due to the requirement of modelling and labelling the data. This research's approach to building comparative heuristics can further be interesting in the context of agent systems: instead of producing strict heuristics for each action, just a comparative one can be made to select the best option, similar to how it is done here for neighborhoods.

GNNs in turn are a growing area in AI, usually leveraged for node and graph classifications. Using them for comparison purposes and comparative binary classification can be generalized in the topic of neighborhood selection, in combinatorial optimization problems that can be represented within graph theory such as VRP and VHP. Additionally, similar techniques can be used in any area concerned with decision-making, where the data can be modelled as a graph and requires classification.

As could be seen in Section 1.4, direct application of GNNs within the solution of a VRP (or another combinatorial optimization problem) is still uncommon and a rather new topic. This project uses neighborhoods with some variation from LNS, which changes to an extent the approach and some of the underlining problems. Some are overcome (e.g, with the current approach neighborhoods are guaranteed to not be represented by big graphs), while others are added (within the smaller neighborhoods, over-smoothing of the represented graph is a constant threat). Overall, however, the method proposed here can be considered in any problem that can be approached with LNS.

5.3.2 Lessons learned

Aside from answering the research questions, there are some additional insights gained through the process of completing this project.

As described in Sections 3.3 and 1.4, GNNs are generally used for classification of graphs, including graph-represented neighborhoods as defined in LNS for combinatorial optimization problems (which, as discussed, in Section 1.3, differ from the neighborhoods in this project). Here, however, it was shown they can be used as a tool for comparing graph representations. In this case, the graph representations need to be tailored to both the type of neighborhood and the problem being solved.

On another note, the graphs in this project are both relatively small and heterogeneous. Such graphs are very susceptible to the problem of over-smoothing, and edges between nodes need to be limited with it in mind. Additionally, when applying GNN on a heterogeneous graph, its architecture needs to accommodate for each node and edge type separately. The processing of each type can alter the learning of the task at hand. Finally, it is important to note that the main power of GNNs is learning of relations and message passing between nodes - thus the information needs to be represented in the correct places (edges or node) and aggregated in a manner that retains their relationships.

Bibliography

[1] 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. *International Journal of Production Research*, 0(0):1–27, 2021.

[2] Marc Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation. *CoRR*, abs/1906.12192, 2019.

[3] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks. *CoRR*, abs/1904.05811, 2019.

[4] Mingxiang Chen, Lei Gao, Qichang Chen, and Zhixin Liu. Dynamic partial removal: A neural network heuristic for large neighborhood search, 2020.

[5] Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020.

[6] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR, 2016.

[7] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[8] Ahmed Douik, Hayssam Dahrouj, Tareq Y Al-Naffouri, and Mohamed-Slim Alouini. A tutorial on clique problems in communications and signal processing. *Proceedings of the IEEE*, 108(4):583–608, 2020.

[9] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P. Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 19–24, 2020.

[10] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[11] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers —& Industrial Engineering*, 57(4):1472–1483, 2009.

[12] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893*, 2019.

[13] Jonas K Falkner, Daniela Thyssens, and Lars Schmidt-Thieme. Large neighborhood search based on neural construction heuristics. *arXiv preprint arXiv:2205.00772*, 2022.

[14] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *CoRR*, abs/1903.02428, 2019.

[15] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *CoRR*, abs/1711.08920, 2017.

[16] Christian M.M. Frey, Alexander Jungwirth, Markus Frey, and Rainer Kolisch. The vehicle routing problem with time windows and flexible delivery locations. *European Journal of Operational Research*, 308(3):1142–1159, 2023.

[17] Lei Gao, Mingxiang Chen, Qichang Chen, Ganzhong Luo, Nuoyi Zhu, and Zhixin Liu. Learn to design the heuristics for vehicle routing problem. *CoRR*, abs/2002.08539, 2020.

[18] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017.

[19] Aric Hagberg and Drew Conway. Networkx: Network analysis with python. *URL: https://networkx. github. io*, 2020.

[20] Ehsan Hajiramezanali, Arman Hasanzadeh, Nick Duffield, Krishna R Narayanan, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks, 2019.

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

[22] André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artificial Intelligence*, 313:103786, 2022.

[23] Jun Hu, Shengsheng Qian, Quan Fang, Youze Wang, Quan Zhao, Huaiwen Zhang, and Changsheng Xu. Efficient graph deep learning in tensorflow with tf_geometric. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 3775–3778, 2021.

[24] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pages 87–104, 2021.

[25] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[26] Guohao Li, Chenxin Xiong, Ali K. Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *CoRR*, abs/2006.07739, 2020.

[27] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

[28] Bo Lin, Bissan Ghaddar, and Jatin Nathwani. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538, 2022.

[29] Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, page 105903, 2022.

[30] Xiaoyu Mo, Yang Xing, and Chen Lv. Heterogeneous edge-enhanced graph attention network for multi-agent trajectory prediction. *CoRR*, abs/2106.07161, 2021.

[31] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[32] Sai Munikoti, Deepesh Agarwal, Laya Das, Mahantesh Halappanavar, and Balasubramaniam Natarajan. Challenges and opportunities in deep reinforcement learning with graph neural networks: A comprehensive review of algorithms and applications. *arXiv preprint arXiv:2206.07922*, 2022.

[33] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[34] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[35] Arslan Ali Syed, Gaponova Irina, and Klaus Bogenberger. Neural network based metaheuristic parameterization, with 1 application to vehicle matching problem in on demand mobility 2 services 3. 2019.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[38] Qi Wang. Varl: a variational autoencoder-based reinforcement learning framework for vehicle routing problems. *Applied Intelligence*, 52(8):8910–8923, 2022.

[39] Yunxuan Xiao, Yanru Qu, Lin Qiu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. Dynamically fused graph network for multi-hop reasoning, 2019.

[40] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

[41] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. *CoRR*, abs/2011.08843, 2020.

[42] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.