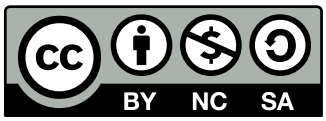
Aalto University
School of Science

Master's programme in Mathematics and Operations Research

Graph Neural Network Heuristic for Timetable Planning in Public Transport

Leevi Rönty

Master's Thesis
2024



Aalto University
School of Science

Author Leevi Rönty

Title Graph Neural Network Heuristic for Timetable Planning in Public Transport

Degree programme Mathematics and Operations Research

Major Systems and Operations Research

Supervisor Asst. Prof. Philine Schiewe

Advisor Asst. Prof. Philine Schiewe

Date 17 March 2024 **Number of pages** 41+1 **Language** English

Abstract

In public transport planning, timetabling is the process of assigning times for vehicle departures and arrivals to stops. Timetable optimization is an important step of a typical public transport planning process. The timetable affects which routes the passengers choose to take. Yet, the choices of routes also affect which timetables are optimal. As such, the best solutions to this problem are obtained when both the timetable and the passenger routes are optimized simultaneously instead of sequentially. However, this quickly becomes computationally intractable. Instead of finding exact solutions, as a heuristic approach, we propose to predict the best passenger routes and then to optimize the timetable using established methods. We propose a graph neural network model for predicting the aggregated passenger counts on the edges of the event activity network. Our approach demonstrates occasional improvements against a shortest path routing baseline. The relationship between the weight prediction loss and the resulting optimality gap indicates, that training the network with a direct regression task may not directly result in improvements of the optimality gap, which is our ultimate objective. Further research in the topic of graph neural networks as a heuristic could explore other training tasks instead of the direct prediction approach. Generating high-quality training datasets is also an area that could be improved.

Keywords Public Transportation Planning, Graph Neural Networks, TimPass, Heuristic

Aalto-yliopisto
Perustieteiden korkeakoulu

Tekijä Leevi Rönty

Työn nimi Graafineuroverkkoheuristiikka joukkoliikenteen aikataulutukseen

Koulutusohjelma Mathematics and Operations Research

Pääaine Systems and Operations Research

Työn valvoja ja ohjaaja Asst. Prof. Philine Schiewe

Päivämäärä 17.3.2024 **Sivumäärä** 41+1 **Kieli** englanti

Tiivistelmä

Aikataulutus joukkoliikennesuunnittelussa on prosessi, jossa määritellään kulkuneuvojen lähtö- ja saapumisajat joukkoliikennepysäkeille. Aikataulun optimointi on tärkeä osa tyypillistä joukkoliikenteen suunnitteluprosessia. Valittu aikataulu vaikuttaa reitteihin, joita matkustajat valitsevat pysäkkien välillä, mutta toisaalta matkustajien suosimat reitit vaikuttavat aikataulun optimointiin. Parhaat ratkaisut aikataulutusongelmaan saadaankin optimoimalla aikataulu ja matkustajien reittivalinnat samanaikaisesti perättäisen optimoinnin sijasta. Tästä tulee kuitenkin nopeasti laskennallisesti haastavaa. Eksaktien ratkaisujen etsimisen sijaan työssä esitetään uusi heuristinen ratkaisumenetelmä, joka ennustaa matkustajien parhaat reitit ykset ja optimoi sitten pelkästään kulkuneuvojen aikataulut. Graafineuroverkkolla ennustamme kokonaismatkustajamääriä graafin kaarille tapahtuma-aktiviteetti -verkossa. Verrattuna lyhyimmän reitityksen vertailukohtaan, menetelmämme tuottaa toisinaan parempia ratkaisuja. Opetustehtävähäviön ja optimaalisuuskuilun välinen riippuvuus ei indikoi, että regressiotehtävän ratkaisu graafineuroverkon opetuksessa kaventaisi suoraan optimaalisuuskuilua, mikä on varsinaisena tavoitteena. Jatkotutkimuksessa koskien graafineuroverkkojen käyttöä heuristiikkana voitaisiin perehtyä verkon muihin opetustehtäviin suoran ennustamisen sijasta. Korkealaatuisen koulutusdatan generointi on myös yksi parannettavista osa-alueista.

Avainsanat Joukkoliikennesuunnittelu, Graafineuroverkot, TimPass, Heuristiikka

Espoo, 17 March 2024

Leevi Rönty

Contents

1 Introduction

Public transportation systems play a role in many people's lives. As such, it is of great public interest to have transportation systems that are both cost effective to operate and offer high-quality services. One perspective on quality is the expected travel time between often-used journey origins and destinations. Typically, the customers of public transport appreciate getting to their destination faster.

The schedule determines when the vehicles arrive and depart from the stops in a public transport system. From the passenger's point of view, an efficient schedule minimizes the expected travel time. However, optimizing for the travel time is not easy. The optimal schedule depends on the routes the passengers choose to get to their destination, but at the same time the chosen routes depend on the schedule. This means, that to guarantee optimal solutions, we must optimize both the schedule and the passenger routes at the same time. This is much more challenging than optimizing the schedule with a pre-determined passenger routing, as the number of routes that each passenger can choose from can be very large, and in a realistic scenario, we will have a large number of passengers. In practice, we are not able to solve this problem optimally for realistically-sized problems.

In this thesis, we develop a heuristic solution method to this integrated optimization problem by trying to predict the optimal routing before solving for the timetable. Good predictions allow us to exclude routing from the optimization problem, making the problem easier to solve. We study if a graph neural network model is able to predict these routes and how the model's predictions will fare against previous heuristics with both small and large public transport networks.

Note, that in this case, we do not differentiate between transportation modes like busses and trains. In general, a real public transportation system would have multiple modes of transportation, but we ignore this for now, as the developed heuristic can be easily extended to a multimodal case.

We begin by reviewing the literature related to the topics of the thesis in Section 1.1. We continue by covering the theoretical background for the problem in Sections 2.1 and 2.2, some simple previous heuristics (Section 2.3), and graph neural networks (Section 2.4). Then we describe the data generation and the experiments in Section 3, after which we conclude by presenting the results (Section 4) and analyzing what we learned in Section 5.

1.1 Literature review

We study the existing literature from three perspectives. First, we explore the existing heuristics, methods, and formulations related to the timetabling and passenger routing problem (TimPass) as seen in [1, 2]. Next, we review the literature for machine learning (ML) methods in public transport optimization, and lastly, we take a look at ML methods in combinatorial optimization.

The TimPass problem as presented in [1] can be viewed as an extension to the periodic event scheduling problem (PESP), originally presented in [3]. In PESP, the objective is to find a peridic timetable for events that minimises the total travel time,

given some fixed routing for the passengers. The TimPass problem extends this by letting the passenger routing be also optimised at the same time as the timetable. This formulation is more realistic by taking passenger behaviour into account, as the passengers typically choose their route based on the timetable and the resulting travel times.

The TimPass problem can be formulated as a satisfiability problem (SAT) [4]. In addition, [4] presents methods for modelling time-varying demand between destinations. Due to the formulation, SAT solvers can be used instead of IP solvers, yielding promising computational results.

The cycle-basis formulation for PESP presented in [5] makes the problem easier to solve. As TimPass is an extension of PESP, the same cycle-basis formulation can also be used in TimPass [2].

Moving on to ML method applications in public transport planning, [6] experiment with predicting the robustness of public transport schedules using multiple statistical ML methods. They are able to predict the robustness accurately apart from a few edge cases which are attributed to the lack of similar training data instances.

Going beyond traditional ML, [7] use deep multi-agent reinforcement Q-learning to optimise the bus timetables in an aperiodic setting. As a case study, they demonstrate that the proposed method is able to reduce the actual operating cost of Beijing's bus network.

Inspired by reinforcement learning (RL) with transformers for solving routing problems ([8]), [9] apply the method to the Transit Network Design and Frequency Setting Problem. They obtain state-of-the-art results for a single benchmark instance.

[10] combine the SAT formulation for PESP with a multi-agent RL scheme, in which they use the agents to generate heuristics for the SAT solver. The results are very promising for a subset of the benchmark instances, hinting that the approach could have more potential if a more complex decision policy was learned.

When it comes to ML heuristics in combinatorial optimization, [11] review the methods on a high level. The authors highlight the diversity of the possible approaches. The existing approaches use either imitation learning or experience-based methods such as RL. However, the algorithmic structure is not tied to only e.g. directly predicting the optimal solution. Some approaches instead learn to configure the underlying solver to perform the best for the problem at hand, while other approaches learn a subroutine that the solvers may use during the solving process, e.g. to learn a cut.

In [12], the review is focused on methods that the solver uses as a subroutine. Like in [11], the authors note the diversity of approaches. Both reviews conclude, that the work on the ML methods in optimization seems promising, but generalisation of the methods to all kinds of optimization problems is yet to be seen.

Continuing with the combinatorial optimization reviews, [13] notes the popularity of graph neural networks (GNNs) in learning the heuristics. They also point out their uses in SAT solvers. In [14], the authors use GNNs to both perform partial assignment of the variables and to guide the branch and bound process of the SCIP solver. The method yields outstanding results for the used large-scale real-world datasets, while also performing well on the MIPLIB benchmark.

2 Methods

In this section, we introduce the used methods and their definitions. First, we present the established mathematical tools used in public transport planning. These include the mathematical representation of public transport systems and the integer programming formulations. Next, we present the graph neural network methods used. Finally, we define how the evaluation scheme ties all these methods together.

2.1 Event activity network data

Before we can define an event activity network, we have to define a few other things.

Definition 1 (Public Transport Network) *A public transport network (PTN) is a simple undirected graph* $PTN = (S, R)$ *with a set of stops* S *and a set of direct connections between the stops* R*.*

A PTN describes the underlying transportation infrastructure of a public transport system. We consider the connections to be undirected. This simplifies the modeling, but one can also have a PTN with directed edges. The PTN can also be non-simple, but that notation would make a difference only if there were either capacity constraints per connection or if multiple modalities were used.

Stops and connections are not enough to model the public transport system. We will use an event activity network (EAN) for that.

Definition 2 (Event Activity Network) *The event activity network EAN is a directed graph* $EAN = (E, A)$ *with a set of events* E *and a set of activities* A *connecting the events. The activities and events are defined later.*

We will define the set of events E and activities A a bit later, as we need the notation of a line concept for that first.

Definition 3 (Periodic timetabling) *A period* $T \in \mathbb{N}$ *defines the time interval at which various events are repeated.*

To contrast with periodic scheduling, in aperiodic scheduling the events do not repeat, but happen only once at the time indicated by the timetable. We are interested in the periodic instead of the aperiodic scheduling problem. This period could be for example 60 minutes, one day, or something else. The appropriate period length depends on what kind of schedule we aim to optimise.

Definition 4 (Line concept) *A line* $l \subset R$ *is a simple directed path in a PTN. A line concept is a set of lines* L *with associated frequencies* $f_l \in \mathbb{N}$ *for all* $l \in L$*. A frequency determines how often a line is served within the period* T*.*

Here, we assume the lines to be directed, meaning that vehicles travel the line only in one direction. Usually, in real-world lines, the vehicles travel in both directions,

but this can also be modeled here by including the reverse direction as a separate line instance.

From the line concept, we can define the set of events E and activities A used in the EAN. For the events, we have the disjoint arrival and departure types: $E = E_{\text{arr}} \cup E_{\text{dep}}$. As the names suggest, the sets E_{arr} and E_{dep} contain the events describing vehicle arrivals and departures at the stops. To simplify the notation, we define the set of available repetition numbers as $R_l = \{1, \ldots, f_l\}$. More formally, the sets are defined as follows:

$$E_{\text{arr}} = \{(\text{arr}, u, l, r) : l \in L, u \in l, r \in R_l\}$$
$$E_{\text{dep}} = \{(\text{dep}, u, l, r) : l \in L, u \in l, r \in R_l\}$$

As seen in the definition, the events consist of the event type, the stop, the line, and the line repetition number which is used to differentiate between separate repetitions of the line in the periodic case.

For the activities, we also have multiple distinct types: drive, wait, change, and sync. Thus, we have that $A = A_{\text{drive}} \cup A_{\text{wait}} \cup A_{\text{change}} \cup A_{\text{sync}}$. The formal definitions are as follows:

$$\begin{aligned}
A_{\text{drive}} =& \{((\text{dep}, u, l, r), (\text{arr}, v, l, r)) : l \in L, (u, v) \in l, r \in R_l\} \\
A_{\text{wait}} =& \{((\text{arr}, u, l, r), (\text{dep}, u, l, r)) : l \in L, u \in l, r \in R_l\} \\
A_{\text{change}} =& \{((\text{arr}, u, l_1, r_1), (\text{dep}, u, l_2, r_2)) : \\
& \qquad (l_1, l_2) \in L^2, l_1 \neq l_2, u \in l_1 \cap l_2, r_1 \in R_{l_1}, r_2 \in R_{l_2}\} \\
A_{\text{sync}} =& \{((t, u, l, r-1), (t, u, l, r)) : (t, u, l, r) \in E, r \geq 2\}
\end{aligned}$$

The drive activities correspond to the driving activity between stops. The activities connect the line's departure events to the corresponding arrival events. The wait activities correspond to the vehicle staying at the station, waiting for the passengers to board and disembark the vehicle. The wait activities connect arrival events to departure events. The drive and wait activities are demonstrated in Fig. 1. The change activities denote the line transfers that the passengers may take. The activities link the arrivals to departures within the same stop that do not belong to the same line. The idea of change activities and multiple lines at the same stop is shown in Fig. 2. The sync activities are slightly different from the other activities, as the passengers can not travel along those edges. For defining the optimization models, we note the set of activities usable for passenger routing as $A_{\text{r}} = A \setminus A_{\text{sync}}$. The purpose of sync activities is to define constraints on the timetable to have some predefined spacing among the repetitions of the lines. How sync activities are connected is shown in Fig. 3. Defining the constraints on all sync activities is not mandatory. In that case, the sync activities without constraints are redundant.

2.2 Integer programming formulations

Next we define the integer programming formulations used. We start with the periodic event scheduling problem (PESP) and its cycle basis formulation. Then, we extend the

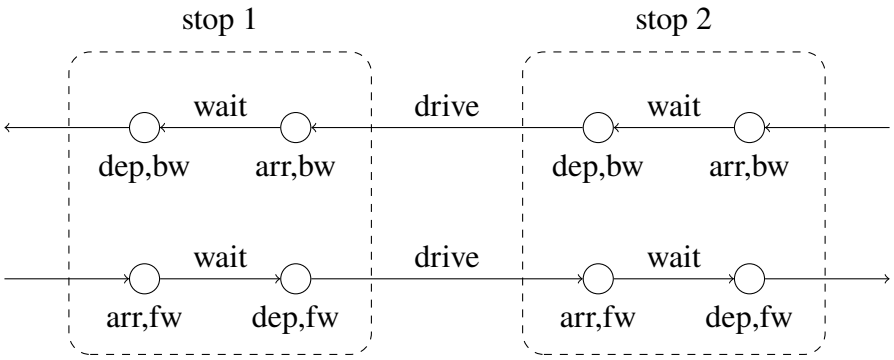


Figure 1: Demonstration for arrival and departure events, drive and wait activities, and line directions. The figure depics how a single line passing through stops 1 and 2 is modelled within the EAN. For both line direction fw and bw we have departure and arrival events at each of the stops. Between the stops, departure events are connected to corresponding arrival events with a drive activity, and within the stop the arrival event connects to a departure event with a wait activity.

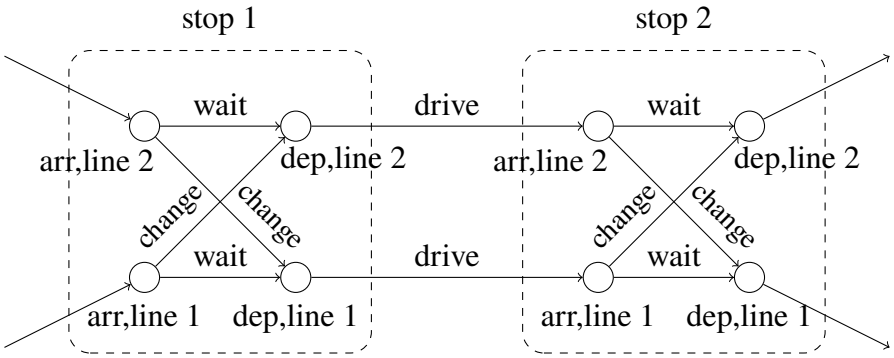


Figure 2: Demonstration for multiple lines at a stop and change activities. We have two lines, that share the stops 1 and 2. Within one stop, the change activities connect arrival events to departure events of different lines. We omit the reverse directions of the lines for clarity in this demonstration.

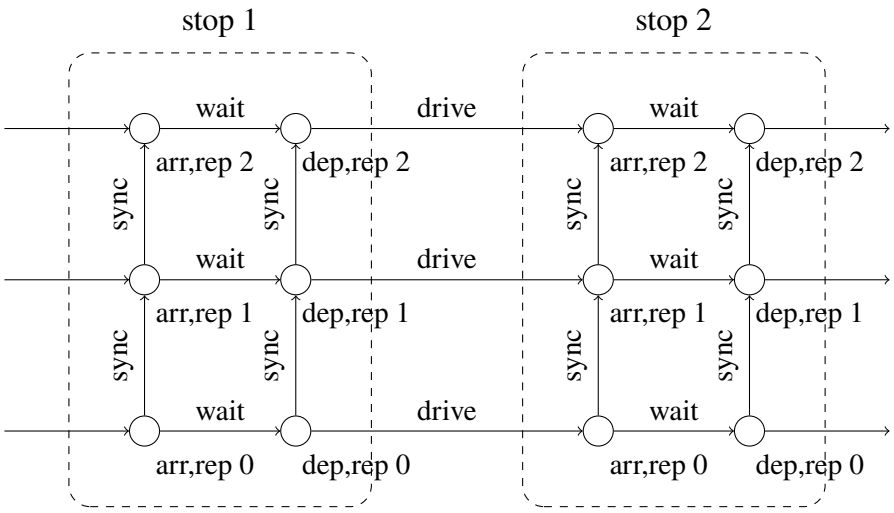


Figure 3: Demonstration for higher frequency lines and sync activities. We have a single line passing through stops 1 and 2, but with multiple repetitions per time period. For eacch repetition, we have a new set of activities and events. For the subsequent repetitions, the events are connected with sync activities. The sync activities are not routable, but are used for constraining the spacing of the repetitions.

PESP problem with passenger routing to obtain the timetabling and passenger routing (TimPass) problem.

2.2.1 Periodic event scheduling problem

The PESP problem is first presented in [3]. We follow the formulation in [2] with notation adapted to be in line with our definition of the EAN. In periodic scheduling, we have a periodic schedule π defined for the events in the EAN. The event times are noted as $\pi_i \in \{0, \ldots, T-1\}$ for all events $i \in E$. As we are working with a periodic schedule, all the event occurrence times can be assumed to be lower than the period of the schedule T.

The activities in the EAN are used to define constraints on the upper and lower bounds for activity durations. For all activities $a \in A$, we have the lower bound $L_a \in \mathbb{N}$ and upper bound $U_a \in \mathbb{N}$. For the bounds, we of course must also have that $0 \leq L_a \leq U_a$. The activity duration for activity $a = (i, j)$ itself is calculated as $(\pi_j - \pi_i - L_a) \bmod T + L_a$. The lower bound term outside of the modulo ensures, that the lower bound is respected. The term inside the modulo is the "slack" time we have due to the schedule not aligning perfectly with the lower bound. As formulated, the duration is guaranteed to be greater or equal to the lower bound. For the given timetable π to be feasible, for all activities $a \in A$ we must have that the upper bound holds: $(\pi_j - \pi_i - L_a) \bmod T + L_a \leq U_a$.

As the modulo operator does not fit well into linear programming as is, we replace the modulo with an integer multiple of the period T. The multiplier $z_a \in \mathbb{Z}$ becomes a decision variable. The lower bounds terms cancel out and the expression for the edge duration is now $\pi_j - \pi_i + z_a T$. As the multiplier z_a can be chosen freely, this is

equivalent to the formulation with the modulo.

In addition to the EAN, we need information on the passenger demand between the stops of the PTN. We note the number of passengers wanting to travel from stop u to stop v as $\mathrm{OD}_{u,v} \in \mathbb{N}$. Not all stop pairs have demand: we express the set of stop pairs with non-zero demand as $\mathrm{OD} = \{(u,v) : (u,v) \in S \times S, \mathrm{OD}_{u,v} > 0\}$.

For routing of the passengers, we assume a fixed routing that is given beforehand. When routing the passengers, each OD pair assumes some path from u to v through the EAN, and for the chosen path edges the weight w_a is incremented by the number of passengers using that route. We will later return to routing in a subsequent section. For now, it's enough to recognize, that we have a weight $w_a \in \mathbb{N}$ for all routable activities $a \in A_\mathrm{r}$ of the EAN.

The total travel time given a timetable is expressed as $\sum_{a=(i,j)\in A_\mathrm{r}} w_a(\pi_j - \pi_i + z_a T)$. In PESP, we try to minimise this by searching for good timetables π. However, when routing the passengers given a timetable, the passengers may not always choose the fastest route, as inconveniences, e.g. transfers, affect the route decision. We model this by including a penalty $b_a > 0$, $a \in A$ that is considered when routing the passengers. In practice, we use shortest path routing but add the penalty to the transfer activity duration.

The total penalised travel time is $\sum_{a=(i,j)\in A_\mathrm{r}} w_a(\pi_j - \pi_i + z_a T + b_a)$. Note, that as w_a and b_a do not depend on π, we can substitute this as the PESP objective without affecting the optimal timetable. The new objective is equal to the total travel time plus a constant $\sum_{a\in A_\mathrm{r}} w_a b_a$. However, we prefer this objective formulation, as it will be directly comparable with the TimPass model objective defined in Eq. (19).

Note, that the objective only considers the routable activities, but the constraints take all activities into account.

$$
\begin{aligned}
(\textbf{PESP}) \quad & \min \sum_{a\in A_\mathrm{r}} w_a(\pi_j - \pi_i + z_a T + b_a) && && (1)\\
\text{s.t.} \quad & L_a \le \pi_j - \pi_i + z_a T \le U_a && a = (i,j) \in A && (2)\\
& \pi_i \in \{0, \dots, T-1\} && i \in E && (3)\\
& z_a \in \mathbb{Z} && a \in A && (4)
\end{aligned}
$$

The formulation for the PESP problem using the schedule π is correct, but it's not the most efficient one available. As presented in [5], instead of explicitly defining the times for the events, we can instead define just the activity durations directly. We note the activity duration as $x_a \in \mathbb{N}$. Now the constraint on activity durations is much simpler: $L_a \le x_a \le U_a \; \forall a \in A$. This formulation also implicitly reduces the number of symmetrical solutions: when directly defining the event times by π we would need to fix some event time explicitly to prevent symmetrical solutions from being considered.

We still need to take the cycles and their feasibility in the EAN into account. In Fig. 4, we have the activities a, b, c, d, e, and f. For the activity durations to be consistent, we must have that the difference of duration of the upper path $x_a + x_b + x_c$ and lower path $x_f + x_e + x_d$ must be an integer multiple of the period T. More formally, $x_a + x_b + x_c - x_d - x_e - x_f = zT$ for some $z \in \mathbb{Z}$. The difference can be a multiple of

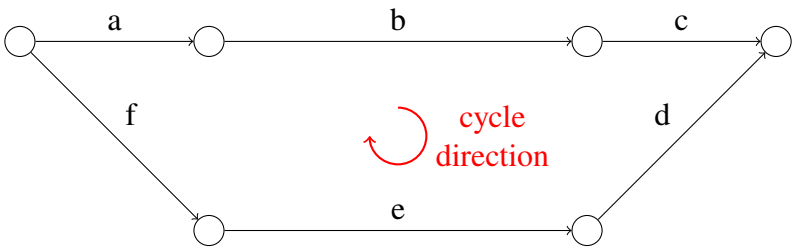


Figure 4: Cycle consistency example.

the period as we are dealing with a periodic timetable, otherwise the difference should be zero.

Definition 5 *(Cycles in a directed graph) A list of activities* $(a_1, a_2, \ldots, a_n)$ *is a simple cycle if each activity is unique:* $a_i \neq a_j, i \neq j$ *for all* i *and* j *in* $\{1, \ldots, n\}$*, adjacent activities share at least one event:* $|a_i \cap a_j| \geq 1$ *for all* $i = 1, \ldots, n$ *and* $j = (i-1) \mod n$*, and the events of the activities are visited exactly once:* $|\cup_{i=1,\ldots,n} a_i| = n$*. The cycle activities are divided into a set of positive edges* c^+ *and negative edges* c^-*. The positive edges are defined as* $c^+ = \{a_i = (e_1, e_2) : e_1 \in a_j, j = (i-1) \mod n\}$*. Likewise,* $c^- = \{a_i : a_i \notin c^+\}$*.*

In the cycle basis formulation, we consider cycles regardless of the edge direction, but the cycle does have a direction. All the edges of a cycle c belong either to the set of "positive edges" c^+ or the set of "negative edges" c^-, based on the direction of the edges along the cycle. In the example of Fig. 4, we have that $c^+ = \{a, b, c\}$ and $c^- = \{d, e, f\}$. In more general terms, for a cycle c to be consistent, for some $z \in \mathbb{Z}$ we must have:

$$\sum_{a \in c^+} x_a - \sum_{a \in c^-} x_a = zT \tag{5}$$

An EAN may have many cycles, but luckily we do not have to check the condition for all of them. Following [5], we can construct a cycle basis for the EAN and checking the condition only for the basis is enough. In short, a cycle basis is a subset of the graph's cycles that can be used to construct every other cycle in the graph. We denote the cycle basis as C. As the cycle inconsistency can be a problem only with activities with constraints, we calculate the cycle basis for activities that can have constraints, i.e. the set A.

Now we may define the cycle basis formulation for the PESP problem. The formulation is quite similar to the original definition, but now the activity duration is expressed in a very concise way and we have the additional cycle consistency

constraint:

$$\textbf{(PESP cycle basis)} \quad \min \sum_{a \in A_{\mathrm{r}}} w_a (x_a + b_a) \tag{6}$$

$$\text{s.t.} \quad z_c T = \sum_{a \in c^+} x_a - \sum_{a \in c^-} x_a \qquad c \in C \tag{7}$$

$$L_a \leq x_a \leq U_a \qquad a \in A \tag{8}$$

$$z_c \in \mathbb{Z} \qquad c \in C \tag{9}$$

$$x_a \in \mathbb{Z} \qquad a \in A \tag{10}$$

2.2.2 Timetabling and passenger routing problem

In the TimPass problem presented in [1], we are free to choose the route used for each OD pair in addition to the timetable as we minimise the total perceived travel time. In PESP, we have fixed passenger counts w_a per activity. In TimPass, we optimise the passenger routing and consequently the weights w_a at the same time as we optimise the timetable. This yields better solutions, as it takes passenger behaviour into account. In general, the passengers prefer to use routes with the shortest travel time to get to their destination.

However, to represent the routes from stop u to stop v in an EAN, we would prefer to have a single predefined node representing the start of paths from stop u and likewise a single node representing the end of the path at stop v. This is contrast to the path starting from or ending at some node in the set of events belonging to a stop. To achieve this, we introduce the auxiliary events E_{aux} and auxiliary activities A_{aux}.

Definition 6 (Auxiliary events and activities) *Each stop $u \in S$ of a PTN is associated with an auxiliary origin and destination event:*

$$E_{aux} = \{(t, u) : t \in \{orig, dest\}, u \in S\}$$

Each auxiliary event is connected via auxiliary activities to the corresponding events in the EAN:

$$\begin{aligned} A_{aux} = &\{((orig, u), i) : i = (t, u, l, r) \in E, t = dep\} \\ &\cup \{(i, (dest, u)) : i = (t, u, l, r) \in E, t = arr\} \end{aligned}$$

The auxiliary event and activity construction is demonstrated in Fig. 5. Now, when considering a route from stop u to v, we know that the path starts from (orig, u) and ends at (dest, v).

To denote the set of all routable activities, we note $A_{\mathrm{r}}^+ = A_{\mathrm{r}} \cup A_{\mathrm{aux}}$. Similarly, to denote all events including the auxiliary events, we note $E^+ = E \cup E_{\mathrm{aux}}$. Auxiliary activities are not considered to be actual activities in the sense of not having a duration. As such, the auxiliary activities are not subject to duration constraints and are not part of the objective function.

To note if the activity a is part of the chosen path between stops u and v, we have the flow variable $p_a^{uv} \in \{0, 1\}, (a, uv) \in A_{\mathrm{r}}^+ \times \mathrm{OD}$. This allows us to set up some

constraints on what is considered a valid path. For a path from stop u to stop v to be valid, it must start at the event (orig, u) and end at (dest, v). Additionally, the path must be connected from the origin all the way to the destination, i.e., for all events that are not auxiliary events we must have an equal number of ingoing and outgoing activities that are used in the path. The activity is used to route from u to v if the corresponding flow variable is one: $p_a^{uv} = 1$.

We can formalise this by introducing a node-arc-incidence matrix $\mathbf{M}$ and an origin-destination vector $q^{uv} = (q_i^{uv})_{i \in E^+}$. For a origin-destination pair $(u, v) \in \text{OD}$, the origin-destination vector elements are:

$$q_i^{uv} = \begin{cases} -1 & \text{if } i = (\text{orig}, u) \\ 1 & \text{if } i = (\text{dest}, v) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

For the node-arc incidence matrix $\mathbf{M} = (m_{i,a})_{(i,a) \in E^+ \times A_\text{r}^+}$, we have that the element is minus one if the activity leads away from the event, one if the activity leads to the event and zero otherwise. Formally:

$$m_{i,a} = \begin{cases} -1 & \text{if } a = (i, j) \text{ for some } j \in E^+ \\ 1 & \text{if } a = (j, i) \text{ for some } j \in E^+ \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

With this notation, we can finally construct the constraint necessary for the set of active flow variables to constitute a valid path from stop u to stop v:

$$\mathbf{M}(p_a^{uv})_{a \in A_\text{r}^+} = q^{uv} \quad uv \in \text{OD} \tag{13}$$

This constraint does not actually prevent the flow variables from creating a cycle, but as we are minimising the total perceived travel time, those cycles would be non-optimal and thus they will not occur in optimal solutions.

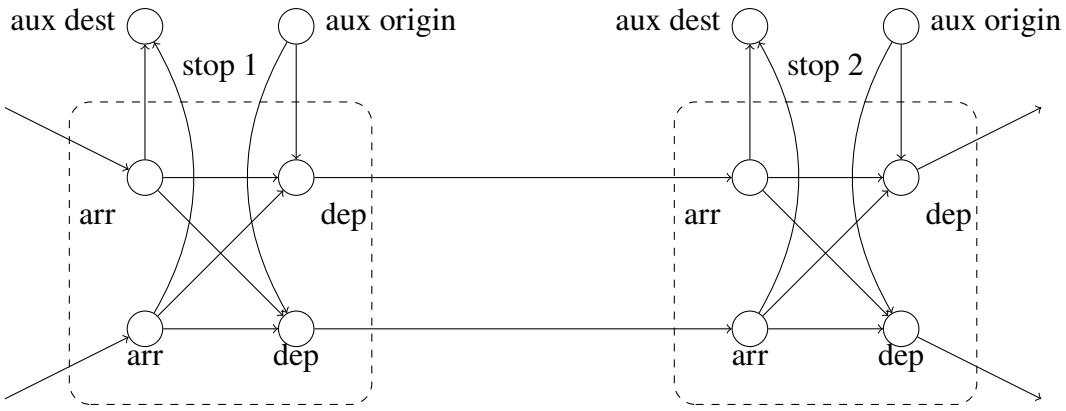


Figure 5: Demonstration of how auxiliary events are connected to the rest of the EAN. Note, that some annotations are left out for clarity. Apart from the auxiliary events and activities, the graph is the same as in Fig. 2.

We can express the total perceived travel time as the sum of perceived travel times over all OD pairs. Thus, we are minimising the following expression:

$$\sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \sum_{a \in A_\mathrm{r}} p_a^{uv}(x_a + b_a) \tag{14}$$

However, we have a problem as we hope to get a linear integer programming problem. Now we have $p_a^{uv} x_a$, which is not linear as both terms are decision variables. Luckily, as the flow variable p_a^{uv} can only be either 0 or 1, we can linearise this expression easily.

We create the linearization decision variable lin_a^{uv} for all $a \in A_\mathrm{r}$ and $uv \in \mathrm{OD}$. We aim to always have $lin_a^{uv} = p_a^{uv}(x_a + b_a)$ without explicitly stating this equality, as the right-hand side has the non-linear term. We achieve this by constructing suitable constraints for the linearization term.

Let B be some large integer for which $B \geq x_a + b_a$ for all $a \in A_\mathrm{r}$. With this value, we can come up with the following set of constraints:

$$lin_a^{uv} \geq 0 \tag{15}$$
$$lin_a^{uv} \leq p_a^{uv} B \tag{16}$$
$$lin_a^{uv} \leq x_a + b_a \tag{17}$$
$$lin_a^{uv} \geq x_a + b_a - (1 - p_a^{uv})B \tag{18}$$

Now, if $p_a^{uv} = 0$, we must have that $lin_a^{uv} = 0$. If $p_a^{uv} = 1$, then $lin_a^{uv} = x_a + b_a$.

Now we can finally express the TimPass problem with the cycle basis formulation. We omit the linearization term and constraints for clarity, but in practice, the linearization must be done for commonly available solvers to function properly.

$$\textbf{(TimPass)} \quad \min \sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \sum_{a \in A_\mathrm{r}} p_a^{uv}(x_a + b_a) \tag{19}$$
$$\text{s.t.} \quad z_c T = \sum_{a \in c^+} x_a - \sum_{a \in c^-} x_a \qquad c \in C \tag{20}$$
$$L_a \leq x_a \leq U_a \qquad a \in A \tag{21}$$
$$\mathbf{M}(p_a^{uv})_{a \in A_\mathrm{r}^+} = q^{uv} \qquad uv \in \mathrm{OD} \tag{22}$$
$$x_a \in \mathbb{Z} \qquad a \in A \tag{23}$$
$$z_c \in \mathbb{Z} \qquad c \in C \tag{24}$$
$$p_a^{uv} \in \{0, 1\} \qquad a \in A_\mathrm{r}^+, uv \in \mathrm{OD} \tag{25}$$

The given formulation for the TimPass problem works, but we are left with many flow variables. Luckily, there is a way to reduce the complexity by calculating which flows can never exist in the optimal solution. This method is first presented in [2].

To formalise the preprocessing, we will need a notation for the shortest paths. In practice, the shortest paths will be calculated with Dijkstra's algorithm.

Definition 7 (Shortest Path) *Let D be the vector of durations for activities so that $D_a \in \mathbb{N}$. We calculate the length of a path P as* $\mathrm{Len}(P, D) = \sum_{a \in P} D_a$. *We define the*

Algorithm 1 Flow variable preprocessing for OD pair (u, v).

Auxiliary event $i = (\text{orig}, u)$
Auxiliary event $j = (\text{dest}, v)$
Initialise $\mathcal{P}_{uv} \leftarrow \emptyset$
Calculate $\beta := len(SP_{i,j}(U^b), U^b)$ ▷ *Longest possible route length from i to j.*
for $k \in E^+$ **do**
    Calculate $\gamma_k := len(SP_{i,k}(L^b), L^b)$ ▷ *Shortest possible path length from i to k*
    Calculate $\delta_k := len(SP_{k,j}(L^b), L^b)$ ▷ *Shortest possible path length from k to j*
for $a = (i, j) \in A^+_r$ **do**
    if $\gamma_i + L^b_a + \delta_j > \beta$ **then**
        $\mathcal{P}_{uv} \leftarrow \mathcal{P}_{uv} \cup \{a\}$ ▷ *The activity can never belong to the shortest path with any timetable, so we ignore it.*

function $SP_{i,j}(D)$ to return a shortest path between the events i and j, i.e.

$$\min_P \ \text{Len}(P, D) = \text{Len}(SP_{i,j}(D), D) \tag{26}$$

Where P is a valid path between events i and j.

The upper and lower bounds can be used as the duration in both Len and *SP*. However, as in the TimPass problem we are dealing with perceived travel times, we should include the penalty term in the duration. Thus, let us define $L^b = (L_a + b_a)_{a \in A}$ and $U^b = (U_a + b_a)_{a \in A}$. Now we can use these to calculate which flow variables p^{uv}_a can never be active in the optimal solution.

In Algorithm 1, we present the algorithm for calculating which flow variables can be safely set to zero. We need to run the algorithm for each OD pair for which we want to reduce the number of flow variables. On a high level, the algorithm first calculates an upper bound as the longest possible route that can belong to an optimal solution. The length of that route under the upper bound edge durations is β. Then, for each activity, we check if the activity can belong to an optimal solution by checking if including the activity would increase the lowest possible perceived travel time for the path above β. In the end, we obtain a set of activities $\mathcal{P}_{uv}$, for which we can set the corresponding flow variable to zero. We handle the preprocessing by including the following constraint in the model:

$$p^{uv}_a = 0 \quad uv \in \text{OD}, a \in \mathcal{P}_{uv} \tag{27}$$

2.3 Shortest path routing heuristic

A simple baseline heuristic to the TimPass problem is to route the passengers according to the lower bound shortest paths and to then solve the timetable with PESP. Formally, we calculate the weights with Algorithm 2 by using the penalised lower bounds L^b as the activity durations. As we do not have capacity constraints on the activities, we can route each OD pair independently of each other.

Algorithm 2 Weight calculation from edge durations

Require: Activity durations D
  $w_a = 0$ for all $a \in A$.
  for $(u, v) \in \mathrm{OD}$ **do**
    $i = (\mathrm{orig}, u)$
    $j = (\mathrm{dest}, v)$
    for $a \in SP_{i,j}(D)$ **do**
      $w_a \leftarrow w_a + \mathrm{OD}_{u,v}$

We use the obtained weights in the PESP to obtain a timetable. The timetable evaluation is conducted in the same way as for the newly proposed heuristic. The evaluation method is described in Section 2.5. Note, that despite the weights used here representing real passengers, we should still do the rerouting as described in the evaluation method, as that can improve the objective. We want to ensure fair evaluation for both the baseline and the proposed heuristic.

2.4 Graph Neural Networks

In general, neural networks are machine learning models with a large number of learnable parameters. The networks consist of multiple additions and multiplications in many layers. Even though the individual operations seem simple, surprisingly complex behaviour can emerge. Graph neural networks (GNNs) are a set of architectures that operate on graph data. To be specific, we focus here on message-passing architectures [15], but other methods for processing graph data do exist.

In GNNs, the graph $G = (V, E)$ contains the node-level features $x_v, v \in V$ and edge-level features $e_{vw}, (vw) \in E$. We note the neighbourhood of node $v \in V$ as $N(v) = \{w : (v, w) \in E\} \subseteq V$. The GNN as in [16] is viewed as a function $f : G \times V \to \mathbb{R}^m$ that maps the node $v \in V$ of a graph G into m-dimensional Euclidian space. Essentially, this generates some $\mathbb{R}^m$ representations for the nodes that we can then use for other downstream tasks, e.g. regression or classification. In [15] the message-passing framework is introduced as a way to formalise the various variants of the graph neural networks. In message-passing GNNs (MPGNNs), the neighbourhoods are used to pass messages between nodes that are used to update the states of the nodes. This differs from the other large class of GNNs that use the spectral representation of graphs [17].

The algorithm contains multiple layers of message passing and processing, which ultimately yields the mapping or embedding of the nodes. We note the number of layers as T. For each layer t, we learn a message construction function M_t and a state updating function U_t. The combination of the functions is used to construct the hidden states of the nodes h_v^{t+1} for the next layer $t + 1$.

The message construction function $M_t(h_v^t, h_w^t, e_{vw}),\ w \in N(v)$ maps the hidden states and the edge features of the neighbourhood of v to messages. These messages can then be aggregated in some edge ordering agnostic way, e.g. a sum in case of the messages belonging to some Euclidean space. The requirement for the aggregation to

be symmetric is important: otherwise the hidden state update would depend on the order in which we process the edges, but this is not justified by the structure of the graph. The messages from the neighbourhood are aggregated to a single aggregated message m_v^t:

$$m_v^t = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{28}$$

The state update function maps the current hidden state and the obtained aggregated message to a new hidden state: $h_v^{t+1} = U_t(h_v^t, m_v^t)$. This forms a layer together with the message generation and aggregation functions. The layers are repeated multiple times, each one with their own trainable parameters.

The obtained node embeddings h_v^T of the last layer are used as inputs for the downstream tasks. For example, in the case of a regression problem, we can have a single-layer perceptron, mapping $h_v^T \mapsto \hat{y} \in \mathbb{R}$. Then, in a supervised setting, we can calculate the loss $\ell(y, \hat{y}) \in \mathbb{R}$ against the true label y and use some optimization algorithm to update the model parameters to minimise the loss.

Typically, an optimization method for neural networks uses some variation of the stochastic gradient descent (SGD). In SGD, the gradient of the loss w.r.t. $\hat{y}$ is propagated back through the network to obtain the gradient with respect to the parameters. In standard SGD, the gradients multiplied by a suitable step size is used as the parameter update. However, it has been noted that this method is susceptible to getting stuck on local minima. In this thesis, we are be using the Adam optimizer [18], which can be viewed as an extension of SGD with momentum and step size adaption.

One desirable quality of MPGNNs is that the same model can be used with networks of varying sizes. This does not limit us to always have the same number of nodes and edges in the problem.

2.4.1 Positional encodings

One limitation of the message-passing framework is, that if the initial node features are equal, some non-isomorphic network structures may not be detected [19]. This can be resolved by injecting some structural information to the features of the nodes. In this case, we will be using Laplacian eigenvector positional encodings (Laplacian PE) [20] to allow the model to better differentiate many kinds of structures.

Let A be the adjacency matrix of a graph. The elements of the Laplacian matrix L are then

$$L_{ij} = \begin{cases} A_{ij}, & i \neq j \\ \sum_{k \in V} A_{ik}, & i = j \end{cases} \tag{29}$$

The Laplacian eigenvectors $\mathbf{f_i}$ are the eigenvectors of the Laplacian matrix. For the positional encoding, we take the eigenvectors with m largest corresponding eigenvalues and inject the resulting vector values to the nodes.

The sign of the eigenvectors can be arbitrary. This is why we choose only top-m vectors, as when training the model we need to pick a random combination of signs.

The model can learn the invariance to the sings symmetries much easier, as we have a limited number of possible combinations.

A naive way of implementing the positional encoding would be to just use some random ordering of the nodes. This has the flaw that we would need to train the model using all possible orderings of the nodes to make the model learn to be invariant to the actual order, and just use it for positional information. As we typically choose $m \ll |V|$, using the Laplacian eigenvectors for positional encoding makes the model training much easier.

We use the Laplacian PE implementation from [21].

2.4.2 Network architecture

The typical MPGNNs are designed for homogenous graphs, meaning that the nodes and edges have the same semantic meaning. However, in this thesis, we need to include also non-homogenous information, as on top of the events and activities we have the OD demands. This pushes us to use more contemporary methods that allow us to model heterogeneous graphs. We chose to use the Heterogenous Graph Transformer (HGT) architecture [22]. The HGT architecture also includes methods for dealing with "web-scale networks" and temporal graphs, but we will omit those methods, as the networks we are dealing with are much smaller and do not vary over time.

In heterogenous graphs $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R})$, the nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$ are associated to node types $\mathcal{A}$ and edge types $\mathcal{R}$ by mapping functions $\tau : \mathcal{V} \to \mathcal{A}, v \mapsto \tau(v)$ and $\phi : \mathcal{E} \to \mathcal{R}, e \mapsto \phi(e)$. This allows us to define the meta-relation $\langle \tau(u), \phi(e), \tau(v) \rangle$ of an edge $e = (u, v)$. This meta-relation is the tuple of the origin and destination node types and the edge relation. Using the meta-relation we can comprehensively state what kind of interaction an edge expresses. This will be used later when defining the message-passing methods of the network. Note, that a node-type pair may have multiple kinds of relations, this is why we need to also include the type of the edge to the meta-relation.

As in the MPGNN architecture, the HGT also consists of multiple layers that generate messages and aggregate them over the neighbourhood of a node v to generate the updated hidden state h_v^t. However, in a heterogeneous setting, the feature distributions of the nodes of different types are also assumed to be different. This motivates the use of different message generation and update functions for different meta-relations and node types. The architecture also uses an attention mechanism to give more weight to messages coming from nodes that the model estimates to be important.

We define the set of all edges from node u to v as $E(u, v)$. Note, that as the node pair may have multiple relation types, $E(u, v)$ is a set with possibly multiple elements,

At a high level, the HGT layer's embedding update function U_t is defined as

$$h_v^{t+1} = U_t(h_v^t, m_v^t) = \text{A-lin}_{\tau(v)}(\sigma(m_v^t)) + h_v^t \tag{30}$$

First, the update functions applies a non-linear transformation σ and then a node-type dependent linear mapping A-lin. The mapped value is then added to the residual connection h_v^t to obtain the updated state. The non-linear transformation used is the

Gaussian Error Linear Unit [23]. All linear mappings *-lin are mappings of form *-lin$(x) = Ax + b$, $x \in \mathbb{R}^m$ with learnable parameters $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$.

The HGT architecture uses multi-head attention when aggregating the messages from various neighbors. Attention is a mechanism that estimates the importance of various messages. The attention values are used to multiply the messages, essentially resulting in a weighted sum aggregation. The multi-head part means, that instead of doing the step above only once, we have multiple attention heads doing the same message-generation and weighted sums in parallel. This allows the model to better pay attention to multiple important inputs.

We note the set of outbound edges from node v and the corresponding neighbors as $NE(v) = \{(u, e) : u \in N(v), e \in E(u, v)\}$. To obtain the aggregated message, we concatenate the weighted sums obtained from the attention heads. The aggregated message m_v^t is calculated as:

$$m_v^t = \sum_{(u,e) \in NE(v)} \Big\|_{i=1}^{n_{heads}} \underset{i,u,e,v}{\text{SoftMax}}(\text{Att-head}^i(u, e, v)) \, \text{MSG-head}^i(u, e, v)$$

The message heads are defined as:

$$\text{MSG-head}^i(u, e, v) = \text{M-Linear}^i_{\tau(u)}(h_u^t) W^{\text{msg}}_{\phi(e)}$$

The message head value is calculated by first calculating a source node type-dependent linear mapping. This is then projected with an edge-type dependent matrix to form the message head.

The attention head yields scalar values that are then normalised with the SoftMax function over the neighbourhood of the node. The attention head is defined as:

$$\text{Att-head}^i(u, e, v) = \text{K-linear}^i_{\tau(u)}(h_u^t) W^{\text{att}}_{\phi(e)} \, \text{Q-linear}^i_{\tau(v)}(h_v^t) \frac{\mu_{u,e,v}}{\sqrt{d}}$$

The attention head first maps the hidden states to a key and a query vector. Typically in attention calculation, we would then directly take the dot product of the vectors. However, this does not take the relation type into consideration. For this reason, we introduce a relation-dependant projection before the dot product. Finally, we scale the result based on a learnable scalar and the number of hidden dimensions d.

2.5 Heuristic evaluation method

The GNN heuristic yields weights for the activities in the EAN and with those weights we solve the schedule with PESP. However, we cannot directly calculate the TimPass objective value from the weights and the schedule, as the weights can be arbitrarily scaled so it does not represent the total number of passengers per activity. We will solve this by calculating the objective minimising properly scaled weights based on the obtained schedule. Luckily, this is easy to do, as the shortest path routing based on the obtained activity durations minimises the objective for the given schedule.

We will now show that the shortest path routing minimises the objective for a fixed schedule. Let $P(u, v)$ be a path between stops u and v. The TimPass objective can formulated equivalently in terms of routes or paths instead of flow variables:

$$\min_P \sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \sum_{a \in P(u,v)} x_a \tag{31}$$

$$= \min_P \sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \, \mathrm{Len}(P(u, v), x) \tag{32}$$

Remember that we don't have capacity constraints for the activities. This means, that the paths are independent of each other. This means, that we can simply minimise each term of the sum independently. By the definition of the shortest path route $SP_{u,v}(x)$, the shortest path routing minimises this objective.

$$= \sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \min_{P(u,v)} \mathrm{Len}(P(u, v), x) \tag{33}$$

$$= \sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \, \mathrm{Len}(SP_{(\mathrm{orig},u),(\mathrm{dest},v)}(x), x) \tag{34}$$

We calculate the objective value for the heuristic from Eq. (34) by setting x to be the activity durations solved from the PESP. This value can be compared with both the shortest path heuristic and the TimPass solution's upper and lower bounds to determine how well the heuristic is doing.

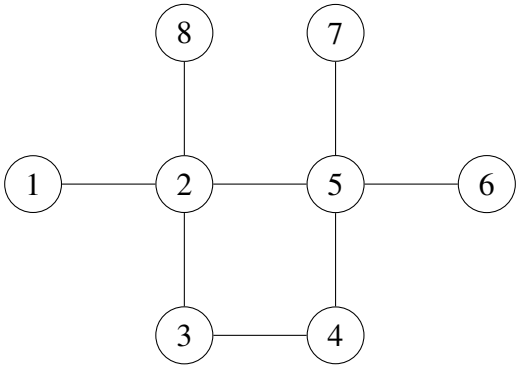


Figure 6: The base PTN on which we generate data.

3 Experiment setup

3.1 Data generation

To train the neural network, we generated a large number of EANs and ODs for which we can solve the TimPass problem to optimality in reasonable time. The problem typically has multiple solutions, but for reasons explained in Section 4.3 we need to control which of the optimal solutions we obtain. All problems use the same PTN with varying lines, duration bounds, frequencies, demands, and penalties. We use a period T of 60 minutes. The PTN is drawn in Fig. 6.

Next we define the notation used for the uniform distribution in the algorithms. We use sampling of distributions to generate variations in the training data.

Notation 1 (Uniform distributions) *By $\mathcal{U}\{x, y\}$ we denote the discrete uniform distribution between $x \in \mathbb{Z}$ and $y \in \mathbb{Z}$, including endpoints. The continuous counterpart $\mathcal{U}[x, y]$ is the continuous uniform distribution between $x \in \mathbb{R}$ and $y \in \mathbb{R}$, excluding y. Note that the bracket type differentiates the continuous distribution from the discrete one.*

First, we list all available lines as the set $\mathcal{L}$ with at least three stops and filter the reversed line versions out. To keep the line plan realistic, we later add the reverse line directions back to the plan. Then, we sample the line plan size $|L| \sim \mathcal{U}\{2, 4\}$. Knowing the size, we sampled the lines belonging to the plan uniformly without replacement from the set of lines: $L \subset \mathcal{L}$. As we want to emulate real EANs, we check whether the resulting EAN would be connected if the reverse directions for the lines were also included. If this is not the case, we sample the line concept L again.

After the set of lines is determined, we sample the line frequencies f_l with the discrete probability mass function $p(f_l) = \sqrt[|L|-1]{f_l/4} - \sqrt[|L|-1]{(f_l - 1)/4}$. The probability mass function is visualised in Fig. 7. The function is designed to give a larger weight for the low line frequencies in case we have only a few lines in total. This regulates the variations in solving times for the generated problems.

As we want the passengers to be able to travel everywhere in the resulting EAN, we also include the lines in the reverse direction. We define the reverse line l_r of

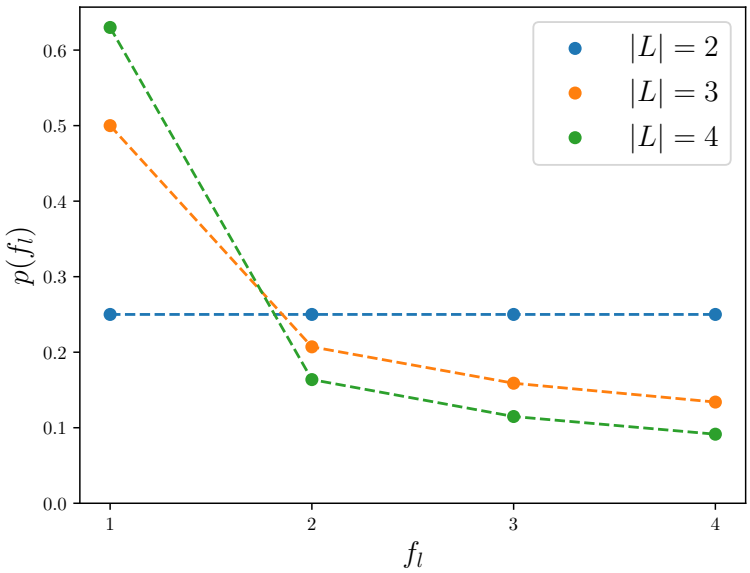


Figure 7: Probability mass functions for the sampling distribution of f_l with various sizes $|L|$.

a line l as $l_r = \{(j,i) : (i,j) \in l\}$. For the EAN, we use the extended line set $L^r = L \cup \{l_r : l \in L\}$. We also set the frequency to the reverse direction to be the same as to the forward direction: $f_{l_r} = f_l$.

From the set of lines and frequencies, we can derive the events and activities of the EAN. We create all events and activities as defined in Section 2.1. For all activities, we also define the upper and lower bounds for the duration. We sample the bounds using Algorithm 3. The algorithm first iterates through lines and relevant stop pairs. After sampling the bounds, the bounds are then set for all relevant activities.

Algorithm 3 Algorithm for sampling the drive activity duration bounds

for $l \in L$ **do**
  for $(u,v) \in l$ **do**
    Sample $\lambda \sim \mathcal{U}\{1,15\}$ ▷ *The lower bound*
    Sample $\omega \sim \mathcal{U}\{0,5\}$ ▷ *Difference between upper and lower bound*
    for $r \in R_l$ **do**
      $a_1 = ((\text{dep}, u, l, r), (\text{arr}, v, l, r))$ ▷ *Drive activity of l*
      $a_2 = ((\text{dep}, v, l_r, r), (\text{arr}, u, l_r, r))$ ▷ *Drive activity of l_r*
      $L_{a_1}, L_{a_2} = \lambda$
      $U_{a_1}, U_{a_2} = \lambda + \omega$

The bounds for wait activities are sampled with the same idea of not having the bounds change between line directions or repetitions. This time we loop over the stops and skip the iteration if the stop is the start or end of the given line. The method is described in Algorithm 4

Algorithm 4 Algorithm for sampling the wait activity duration bounds

for $l \in L$ **do**
  for $s \in l$ **do**
    if s is the start or end of l **then**
      Continue
    Sample $\lambda \sim \mathcal{U}\{1, 3\}$ ▷ *The lower bound*
    Sample $\omega \sim \mathcal{U}\{0, 2\}$ ▷ *Difference between upper and lower bound*
    for $r \in R_l$ **do**
      $a_1 = ((\text{arr}, u, l, r), (\text{dep}, u, l, r))$ ▷ *Wait activity of l*
      $a_2 = ((\text{arr}, u, l_r, r), (\text{dep}, u, l_r, r))$ ▷ *Wait activity of l_r*
      $L_{a_1}, L_{a_2} = \lambda$
      $U_{a_1}, U_{a_2} = \lambda + \omega$

For the change activities, the sampling process is a bit different. First, we sample the lower bound $\lambda_u \sim \mathcal{U}\{1, 5\}$ for all $u \in S$. Then, $L_a = \lambda_u$, $U_a = \lambda_u + T - 1$ for all activities $a = ((\text{arr}, u, l_1, r_1), (\text{dep}, u, l_2, r_2)) \in A_{\text{change}}$. In this case, the difference between the upper and lower bound is always $T - 1$. This ensures, that the change is always feasible. We have the same bounds for all transfers that happen at the same stop.

By the definition used here, we can have different penalties for all the change activities, and nothing limits us to having penalties only for those activities either. However, to keep things simple and aligned with other datasets, we just sample $\rho \sim \mathcal{U}\{1, 5\}$ and set $b_a = \rho$ for all $a \in A_{\text{change}}$ and $b_a = 0$ otherwise.

For sync activities, we don't sample the bounds. Instead, we simply set

$$L_a, U_a = T / f_l \quad \forall a = ((t, u, l, r_1), (t, u, l, r_1)) \in A_{\text{sync}} \tag{35}$$

Note, that $L_a, U_a \in \mathbb{N}$ as we chose T and f_l sampling to enforce this. This is necessary, as the TimPass model formulation is defined in terms of integers.

Finally, we must come up with an OD matrix for the problem instance. Only the stops that belong to a line are present in the EAN. As the lines are randomly chosen, the set of stops is also random. To both keep the number of OD pairs feasible and to introduce variations in the data, we limit the number of OD pairs considered. When $|S| \geq 7$, the problem has at least $7 \cdot 6 = 42$ possible pairs. Instead of using them all, we take a random subset of the possible OD pairs. We sample $|\text{OD}| \sim \mathcal{U}\{30, 40\}$ and then take the random subset $\text{OD} \subset S^2$. If we did not sample $|\text{OD}|$, we just use all possible OD-pairs. For all OD-pairs we sample the demand as $\text{OD}_{uv} \sim \mathcal{U}\{1, 20\}$ for all $uv \in \text{OD}$.

Now we can define the TimPass problem and attempt to solve it. We try to solve the obtained problem with Gurobi with a time limit of 20 seconds. If the optimal solution is not obtained within this limit, we deem the problem difficult to solve and try sampling the problem again. This happened in approximately 48% of the sampled problem variations. In Fig. 8 we present the shares of the node set sizes for the attempted instances and the solved instances. After discarding difficult problem instances, the

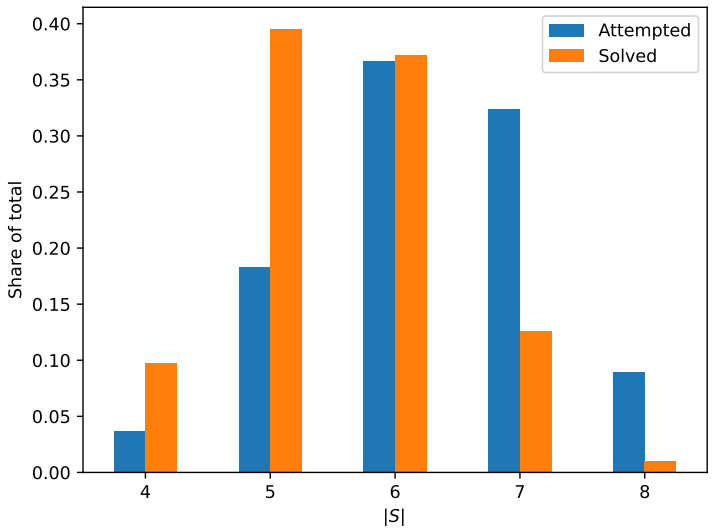


Figure 8: The node set sizes of the generated instances before and after of discarding difficult problems.

smaller problems are more prevalent. This means, that the larger problems were more often not solved within the time limit.

As stated previously, we need to be able to control which of the multiple solutions we actually get. This is done by first solving the problem as usual. We denote the calculated optimal objective value V^*. To obtain other solutions, we add a new constraint and change the objective function. The new constraint is that the objective must be equal to V^*:

$$\sum_{uv \in \mathrm{OD}} \mathrm{OD}_{uv} \sum_{a \in A_\mathrm{r}} p_a^{uv}(x_a + b_a) = V^* \tag{36}$$

The idea for using the previous objective value as a constraint comes from [24].

For the new objective, we change it to be the weighted sum of the flow variables. The weights are sampled randomly between zero and one:

$$\min \sum_{uv \in \mathrm{OD}} \sum_{a \in A_\mathrm{r}} w_a^{uv} p_a^{uv}(x_a + b_a), \quad w_a^{uv} \sim \mathcal{U}[0, 1] \tag{37}$$

Assuming that we can not sample the same weight for multiple flow variables, this ensures that the modified problem has only one optimal solution.

With the problem given above, we attempt to generate 10 solutions for each problem instance. We only keep the unique solutions. Then we write the solutions to disk to be used later for the neural network training and evaluation.

3.2 Data representation as a heterogenous graph

On top of the EAN data, we have the OD data we need to include as input to the GNN. We use a heterogenous graph for this, as it allows us to include multiple kinds of nodes

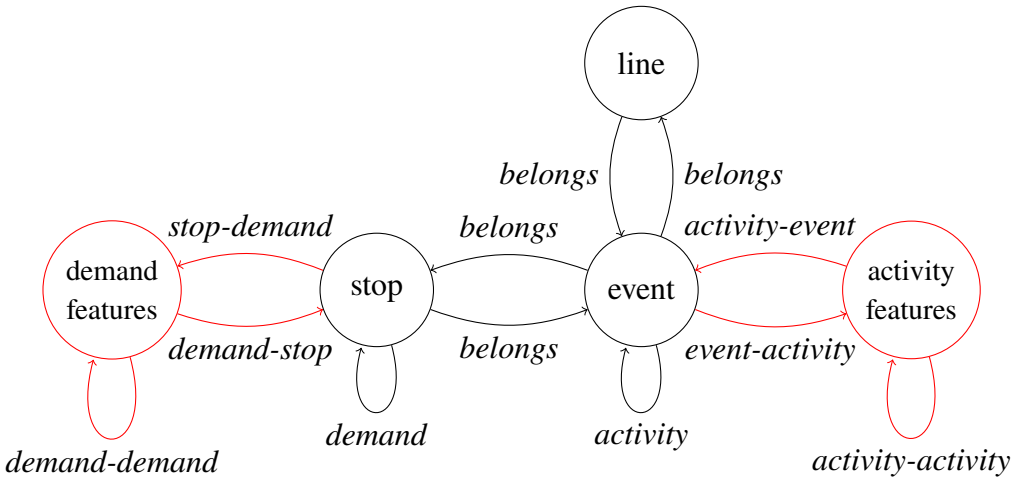


Figure 9: The node and relation types of the heterogenous representation after the line-graph transformation. The node types and relations added by the transformation are marked in red.

and relations in a graph. The simplified overview of the node types and their relations is expressed in Fig. 9.

The set of node types is $\mathcal{A} = \{\text{event}, \text{stop}, \text{line}\}$. As the line and stop ids are essentially meaningless on their own, we chose to represent the identity of a line and a stop as a separate node in the graph. This allows us to meaningfully represent which events belong to which stop and line without needing to e.g. permute the ids when training the model. The node type defines what kinds of features the node has. The line and stop type nodes do not have any features. The event-type nodes have one-hot encoded the event type, either arrival or departure.

The set of edge types is $\mathcal{R} = \{\text{activity}, \text{demand}, \text{belongs}\}$. The activity-type edges have the activity type one-hot encoded. The activity duration lower bound L_a, upper bound U_a, and penalty b_a are included as a share of the period T. The normalised shortest path routing weight is also included. In the preprocessing step we deduced that some of the flow variables must be zero. We chose to not encode this information as is, but as an aggregate of the share of flow variables that were preprocessed. We controlled for the optimal solution we obtained from the TimPass model with the preference variables. To keep the regression target disambiguous, the preferences are also included in the features. The demand edges store the number of customers wanting to travel between the stops. The value is normalised to have a maximum value of one across all edges. The belongs edge type does not contain any features.

Unfortunately, the implementation of HGT we are using does not use edge features directly. Instead, we encode the edge features by connecting the line graph representation of the edges to the rest of the graph. A similar trick is used in [25]. The method is visualised in Fig. 10. We applied the line-graph conversion one edge type at a time for the activity and demand edge types. Other edges were left untouched.

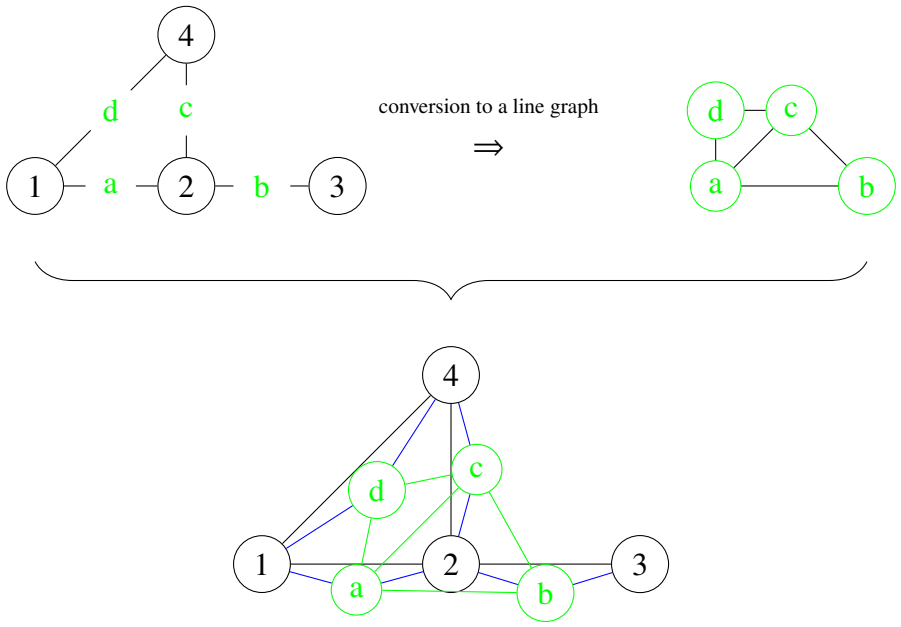


Figure 10: Demonstration of how the line graph trick is used to include edge features in the graph. The original graph with edge features is top left. The line graph on top right turns edges into nodes and connects the new nodes that were previously connected to a same node. The original graph and the line graph are then joined by connecting the edge nodes to corresponding original nodes, depicted by the blue edges. The method is described in detail in Algorithm 5.

3.3 Training

We train the GNN model to predict the normalised weights w of the optimal passenger routing. The weights are normalised to have a mean of one, as having a consistent scale should help the model. The model parameter updates are done to minimise the l^2 loss of the predictions $\hat{w}$ against the know optimal weights w^*. The parameter updates are done using the Adam algorithm [18].

The training process and the model have multiple hyperparameters governing the model architecture and parameter updates. Typically, the hyperparameter choices need to be done carefully, as not all hyperparameter choices will yield good results. Hyperparameter optimization is the process of optimizing the hyperparameters itself instead of the model parameters, so that the combination of hyperparameters would

Algorithm 5 Line graph extension

Require: Graph $G = (V, E)$

$E_{\text{line}} = \{(e, f) : (e, f) \in E^2, e \cap f \neq \emptyset\}$ ▷ *The edge pairs that share an endpoint*

$L(G) = (E, E_{\text{line}})$ ▷ *The line graph*

$V_{\text{joined}} = V \cup E$

$E_{\text{joined}} = E \cup E_{\text{line}} \cup \{(v, (v, n)) : v \in V, n \in N(v)\}$

Joined graph $J = (V_{\text{joined}}, E_{\text{joined}})$

yield good results in the training process. We choose to use a Bayesian search for this [26]. The hyperparameters we aim to optimize are the learning rate γ, the layer count T, the hidden latent dimension d, and the number of attention heads n_{heads}.

In Appendix A we present the results for the hyperparameter optimization and the hardware used. The computations are performed using resources within the Aalto University School of Science "Science-IT" project.

4 Results

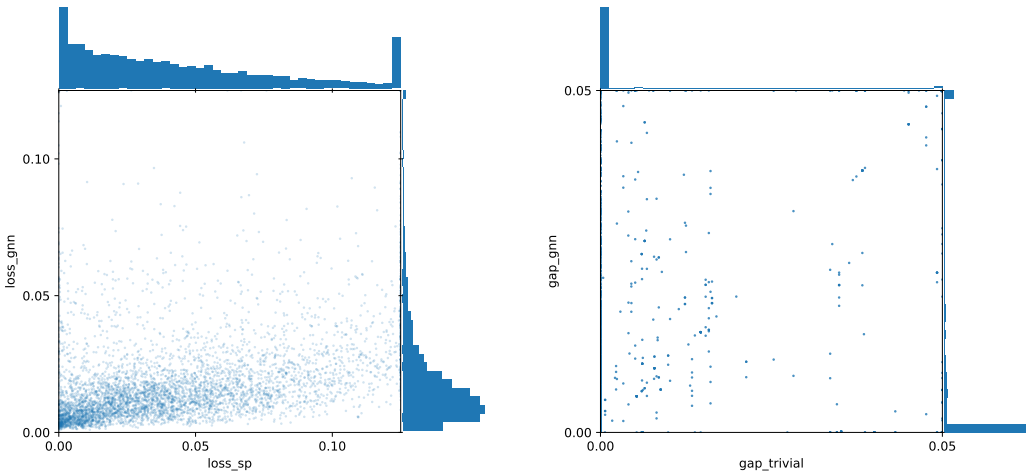
4.1 Heuristic performance

We calculated the performance metrics on a test dataset with over 5000 samples. This dataset was not used in the training of the GNN heuristic.

For each problem instance, we calculated the GNN heuristic loss (as MSE against the optimal activity weights) and optimality gap as described in Section 2.5. We also calculated the metrics for the shortest path (SP) heuristic weights. First, in Fig. 11 we compare the joint distributions of the losses and optimality gaps of the heuristic against each other.

For the calculated losses in Fig. 11a we can clearly see, that the GNN heuristic typically achieves lower losses than the SP heuristic. This indicates that the trained network did at least partially work as intended, as the heuristic was trained to minimise the loss. The loss is smaller for the GNN in 79% of the problem instances.

However, observing the results in Fig. 11b is more difficult. For the SP heuristic, the gap is zero in 91% of the cases and for the GNN heuristic in 61%. Notably, the



(a) Shortest path heuristic loss vs GNN loss. **(b)** Shortest path heuristic gap vs GNN gap.

Figure 11: Join distributions for the shortest path heuristic and GNN performance metrics. Marginal distributions as histograms along the edges. The views are clipped slightly and the clipped values are collected in the last bins of the histograms.

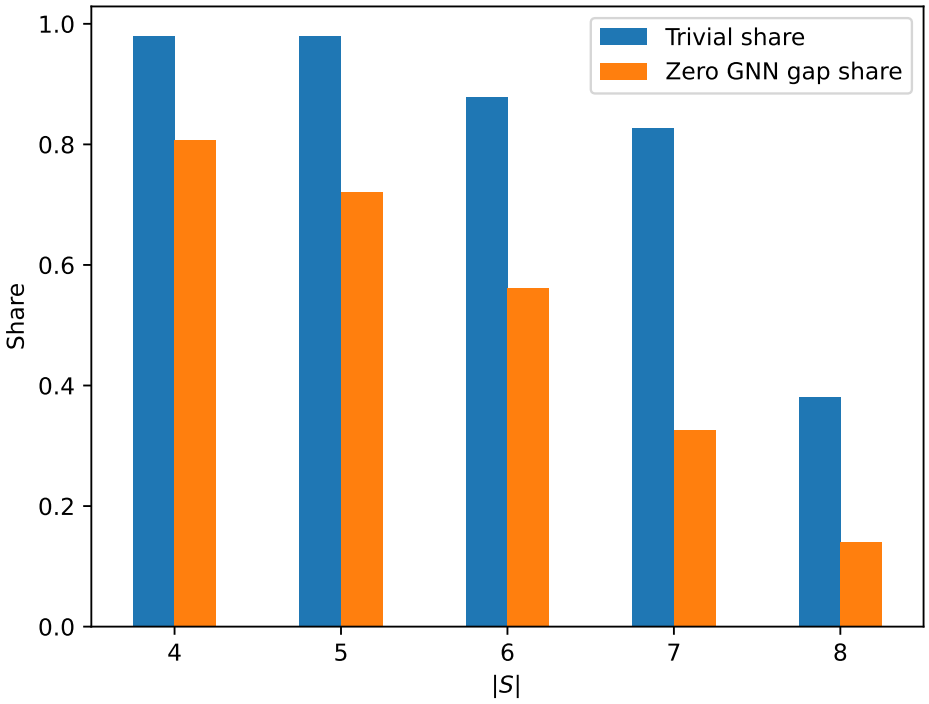


Figure 12: The share of trivial problem instances and zero GNN optimality gaps by instance size.

SP heuristic finds the optimal solution more often than the GNN heuristic despite having larger losses. More about that in Section 4.2. However, we now restrict our comparison to only cases where the SP heuristic optimality gap is nonzero. As we are interested in improving the found solutions, we limit our focus only to cases where we know beforehand that improvements are possible. We use the name trivial problem for the problems where the shortest path heuristic achieves zero optimality gap.

In Fig. 12 we break down the share of trivial problems by the instance size. We note, that for problems with four or five stops, the instances are almost always (98% of cases) trivial. This share decreases to only 38% at eight stops. The share of zero GNN gap instances is always lower than the share of trivial instances.

In Fig. 13 we represent the optimality gap joint distribution without the trivial problems. Note, how the histograms are no longer concentrated around zero. With this filtering, the GNN heuristic achieves a better gap in 41% of the problems. Likewise, the SP heuristic has a better gap for 47% of the problems. When the solutions differ, the difference is often quite large: if GNN gap is smaller, on average it is only about 31% of the SP heuristic gap. This means, that the proposed heuristic can find solutions that are vastly better than the baseline shortest path heuristic. However, this is not consistent.

In Table 1 we present the calculated heuristic performance metrics in tabular form.

We also evaluated the GNN and SP heuristics on selected TimPassLib instances.

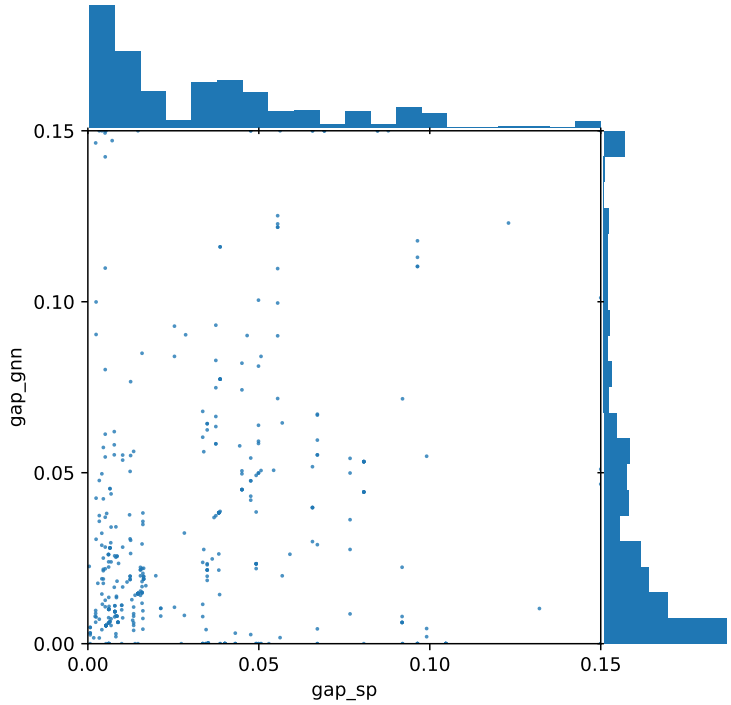


Figure 13: Optimality gap joint distribution for SP and GNN heuristics. Excluding trivial problems.

In Table 2 we present summary statistics comparing the validation dataset instances to the selected benchmark instances. Note, that the validation instances are drawn from the same distribution as the training instances. We chose the benchmark instances to represent a reasonable range of problem sizes. The benchmark instances are much larger and more diverse than the training problems. This allows us to probe the generalization ability and scalability of our heuristic approach. We compare the GNN heuristic results against the SP heuristic and the best published lower and upper

Table 1: Heuristic performance metrics both before and after filtering for trivial problem instances.

	All problems		Non-trivial problems	
Metric	GNN heuristic	SP heuristic	GNN heuristic	SP heuristic
Loss mean	**0.018526**	0.045599	**0.016793**	0.032772
Loss median	**0.013557**	0.034702	**0.013480**	0.022635
Gap mean	0.011484	**0.002880**	0.036789	**0.034759**
Gap median	**0.000000**	**0.000000**	**0.019865**	0.025327
Loss smaller share	**0.7851**	0.2149	**0.7210**	0.2790
Gap smaller share	0.0349	**0.3532**	0.4066	**0.4657**
Average relative gap improvement			0.31	

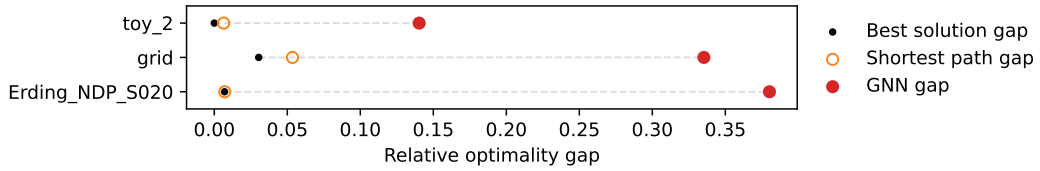


Figure 14: Calculated GNN and SP heuristic optimality gaps against the best published upper bound for the TimPassLib instance.

Table 2: Counts of stops, lines, events, activities, and OD pairs for the datasets used in the evaluation. Using averages for the validation data as there are multiple instances of validation problems.

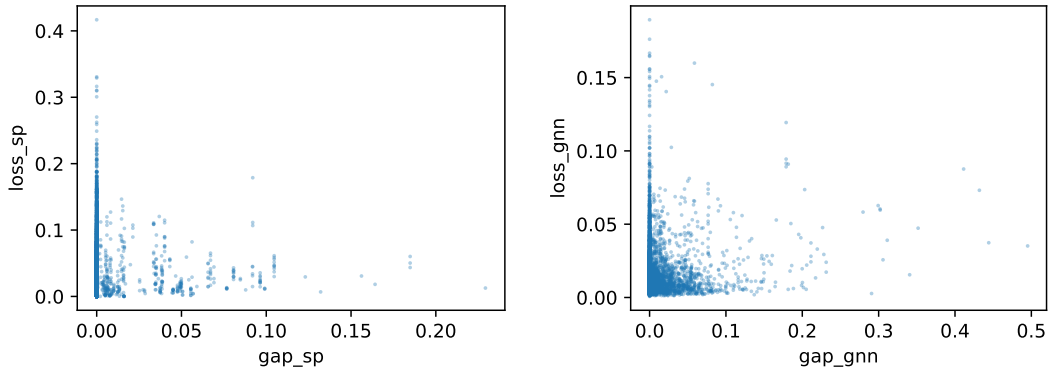
Dataset	Stops	Lines	Events	Activities	OD pairs
Validation instances (average)	5.5	2.5	55.6	150.2	24.9
toy_2	8	6	156	1088	46
grid	25	8	382	2382	567
Erding_NDP_S020	51	21	1132	5300	675

bounds. The PESP time limit was set to 6 hours for the evaluation. The evaluation was performed on a machine with CPU 8 cores and 8 GB of memory. Regarding the PESP computation time, all the SP heuristic results were obtained in less than 11 seconds, while the GNN heuristic for toy_2 took 2h 29min, and the grid and Erding_NDP_S020 hit the time limit of 6 hours. The visualised results are represented in Fig. 14. As observed from the figure, the GNN heuristic yields much greater optimality gaps than the SP heuristic, regardless of the instance size. This hints, that the new method does not generalize well, as is a common limitation of many other ML approaches in combinatorial optimization ([13]).

4.2 Relation of loss and heuristic optimality gap

The chosen GNN heuristic training has the inbuilt assumption, that achieving good predictions for the optimal weights w leads to low optimality gaps. However, after observing the achieved losses and optimality gaps for both the GNN and the SP heuristic, this does not seem to be so straightforward.

In Fig. 15 we observe the loss vs optimality gap plots for both heuristics. If the assumption was correct, we would observe some kind of positive correlation between the metrics. However, the metrics are virtually independent of each other. Only when we focus on very small losses as in Fig. 16 we see any dependence. For very small losses the optimality gaps tend to also be very small.



(a) Gap vs loss for the SP heuristic.

(b) Gap vs loss for the GNN heuristic.

Figure 15: Relationship between the heuristic loss and the achieved optimality gap.

4.3 Theoretical loss lower bound without preference of solutions

We will next investigate what would be the theoretically lowest obtainable loss if we did not include the preference information for the flow variables. As observed, for a given problem instance, we may have multiple optimal solutions to the TimPass problem. We assume that the optimal solution which we obtain is uniformly random between all optimal solutions if we do not control for this with the preference mechanism.

The neural network is trained to minimise the expected average squared error of

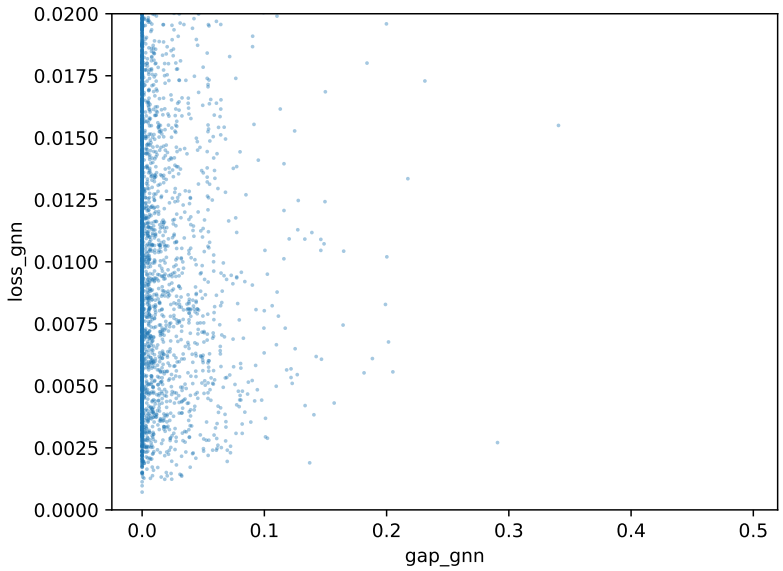


Figure 16: Zoomed view to Fig. 15b. We observe, that for very low loss values the gap tends to also be zero.

the weight predictions. The loss for an individual problem is defined as $\ell(\hat{w}, w) = 1/|A| \sum_{a \in A} (\hat{w}_a - w_a)^2$. If we don't consider preference information, then w can be viewed as a random variable. In this case, the optimal predictor would predict $\hat{w}_a^* = \mathbb{E}[w_a]$. By substituting this back to the loss formula, we get that the lowest obtainable expected loss is the mean population variance of the weights, with the population here being the set of optimal solutions for one problem instance and the average being taken over all problem instances.

We can estimate this lowest obtainable loss by calculating the sample variances for the weights. With this method, we get that $\mathbb{E}[\ell(\hat{w}^*, w)] \approx 0.0091$. As this is lower than the best loss achieved with the preferences, we can deduce that the lack of preference information would not have limited the performance of the model significantly.

5 Discussion

As an overview of the obtained results, the GNN heuristic does not bring a consistent improvement against the SP heuristic. On non-trivial problems sampled from the same distribution as the training data, the new method may improve the results, but whether improvements are possible can not be known beforehand.

The instances in TimPassLib represent a different data distribution than the training data. By testing the GNN heuristic against the benchmark instances, we found that the GNN heuristic does not generalise well to distributions outside of the training data. One of the reasons we chose the GNN architecture for our heuristic was the technical feasibility of using the same network for problem instances of different sizes. As the capability to generalise was very weak, we did not manage to observe if the problem instance size plays a role in performance. The possible performance degradation due to scaling the instances was masked by the large optimality gaps due to generalisation issues.

Another area for improvement is the training dataset we used to train the model. We did have a large number of training instances, but the instances were not very diverse: most of the instances were trivial in the sense that the SP heuristic was optimal, and the base PTN was shared between all instances. It is not clear if the model learns anything useful when introduced to the trivial problems, as the SP heuristic weights are also included in the training data. Not being introduced to various PTNs could hinder the generalisation to other datasets. Due to practical concerns regarding the computational complexity of the TimPass problem, we were forced to limit the computation time when generating the training samples. This could introduce bias, as the training examples are very hard to solve by existing methods while generating heuristics for difficult problems is ultimately our goal.

As we saw in Section 4, at least for the training dataset the SP heuristic was often very good. On the other hand, when the SP heuristic failed, the GNN heuristic was at times able to find improvements. Combining these heuristics to a joint heuristic where the GNN prediction would be used in cases where the SP weights are predicted to be non-optimal could thus yield solutions that are better than what a singe heuristic can achieve. One approach to predicting where the SP heuristic may fail is to investigate the tightness of the bounds for the edge durations. This way we could classify the OD pairs to two groups by the potential for improvement. Training the GNN heuristic only on the high-potential OD pairs could in part alleviate the issue of lack of variation in the training data.

Lastly, as this thesis was focused on investigating the idea of using a GNN heuristic, we did not invest in incorporating the latest solving methods and tricks for the TimPass and PESP problems. As noted earlier, the time limitation on the TimPass problem could introduce some bias. As we are required to solve the PESP problem in the evaluation phase, introducing the best available solution methods can possibly help us evaluate the large-scale problems more accurately.

The question remains if predicting the weights is a good idea in the first place. As we compared the optimality gaps against the l^2 loss on the weight prediction in Section 4.2, we did not observe any strong correlation between the values. This

indicates, that training the model to predict the weight does not necessarily result in a lower gap against the reference solution. Additionally, as the allowed predictions were continuous, sometimes the low but non-zero weight values caused the subsequent PESP solving to be considerably slower than solving the PESP with the SP weights.

To get around these issues, we came up with two ideas for further research into the topic of using a NN heuristics for the routing part of the TimPass problem. First, instead of training the model to predict the weights, we could instead try to predict the routes. This would make sure, that the resulting weights are consistent with the passenger demands, allowing us to skip the SP rerouting in the evaluation phase. The second idea is to try to minimise the TimPass objective value itself by means of reinforcement learning. As improvements in the regression task do not directly translate to better objective values, we could try to to minimise the objective instead. However, RL training is much more difficult to get right than the simple supervised regression task.

Another point of discussion is the suitability of the HGT architecture for this problem. The family of convolutional graph networks are well-suited for tasks with local dependencies. This is also reflected in the typical benchmarks, e.g. the Open Academic Graph link prediction [27], on which the solutions do not depend on long-range interactions. However, the task of routing the passengers optimally depends on global interactions: a small difference in the network, e.g. a high lower bound for an edge, may have large consequences far away, as now it is optimal to route the passengers with a completely different path to their destination.

GNNs also have some well-known limitations. The message-passing architecture is susceptible to over-squashing of information, as the nodes become a bottleneck for the messages [28]. This hinders the architecture's performance on tasks that depend on long-range interactions. The attention mechanism does help with this issue.

The mechanism of passing messages within the one-hop neighbourhood acts as an inductive bias. It guides the model towards discovering short-range patters and as such, this mechanism may not be well-suited for tasks involving long-range interactions. Some recent research directions involve omiting this bias by letting the nodes communicate globally, essentially creating a transformer for graphs [29, 30]. The resulting models have achieved good results on e.g. the quantum-chemical regression task on the OGB-LSC PCQM4Mv2 dataset, which depends on global interactions [31].

6 Summary

In this thesis, we study a novel graph neural network-based heuristic for the timetabling and passenger routing problem (TimPass). Heuristical approaches to the problem are interesting, as including routing makes the problem more realistic but also much more challenging to solve optimally. Improvements to solutions for large-scale problems can be helpful in public transport planning and by extension lower the transportation costs for everyone involved.

We conduct a literary review focused on recent advances and research directions in timetabling for public transportation, machine learning applications in public transport planning, and how neural networks have been used to support solving combinatorial optimization (CO) problems. We found, that the proposed heuristic has not been studied before, and that graph neural networks have seen popularity in CO applications.

We present the methodological foundation for the event activity network representation, optimization problem formulation, and the heuristic model architecture. The practical problem representation issues of connecting the novel heuristic to the established formulation are also discussed and the workarounds are demonstrated. As the heuristic output by itself is not restricted to correspond to the passenger counts, we also devise a process for consistently evaluating the weight predictions.

In the results section, we demonstrate how the new heuristic performed with various problem instance types. First, we examine the performance on the validation data generated by the same data generation process as for the training data. Second, we apply the heuristic to larger and more varied benchmark instances and compare the results to the shortest path heuristic solutions and the published best solutions of the problem. We observe, that at times the new heuristic is able to yield improvements over the SP heuristic, but this is mostly limited to the distribution of instances the model is trained with. With the benchmark instances, the model does not seem to generalise well to the new distribution. The results on the impact of the instance size on performance are inconclusive, as no generalization was observed at all.

Finally, we discuss the limitations and caveats related to the new method and the training process. We identify multiple possible reasons for the poor performance, starting from the relevance of the regression task itself and ending with contemplation on the architecture choice and recent advancements in the field of machine learning with graphs. Lastly, we propose a method of splitting the workload between two heuristics to possibly get the best of the two approaches in one method.

To conclude, we explored heuristical approaches for the TimPass problem and investigated the capabilities of the proposed novel heuristic while critically commenting on the choice of methodology. In the process, we learned a lot about how graph neural networks could be applied to the problem and proposed multiple promising avenues for future research. Our first attempt at cracking the problem was not a complete success, but after all, research is an iterative process.

References

[1] M. E. Schmidt *et al.*, *Integrating routing decisions in public transportation problems*. Springer, 2014.

[2] P. Schiewe and A. Schöbel, "Periodic timetabling with integrated routing: Toward applicable approaches," *Transportation Science*, vol. 54, no. 6, pp. 1714–1731, 2020.

[3] P. Serafini and W. Ukovich, "A mathematical model for periodic scheduling problems," *SIAM Journal on Discrete Mathematics*, vol. 2, no. 4, pp. 550–581, 1989.

[4] P. Gattermann, P. Großmann, K. Nachtigall, and A. Schöbel, "Integrating passengers' routes in periodic timetabling: a sat approach," in *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[5] L. Peeters and L. Kroon, "A cycle based optimization model for the cyclic railway timetabling problem," in *Computer-aided scheduling of public transport*. Springer, 2001, pp. 275–296.

[6] M. Müller-Hannemann, R. Rückert, A. Schiewe, and A. Schöbel, "Estimating the robustness of public transport schedules using machine learning," *Transportation Research Part C: Emerging Technologies*, vol. 137, p. 103566, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X22000146

[7] H. Yan, Z. Cui, X. Chen, and X. Ma, "Distributed multiagent deep reinforcement learning for multiline dynamic bus timetable optimization," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 1, pp. 469–479, 2022.

[8] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[9] A. Darwish, M. Khalil, and K. Badawi, "Optimising public bus transit networks using deep reinforcement learning," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[10] G. P. Matos, L. M. Albino, R. L. Saldanha, and E. M. Morgado, "Solving periodic timetabling problems with sat and machine learning," *Public Transport*, vol. 13, no. 3, pp. 625–648, 2021.

[11] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377221720306895

[12] J. Zhang, C. Liu, X. Li, H.-L. Zhen, M. Yuan, Y. Li, and J. Yan, "A survey for solving mixed integer programming via machine learning," *Neurocomputing*, vol. 519, pp. 205–217, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222014035

[13] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 130, pp. 1–61, 2023.

[14] V. Nair, S. Bartunov, F. Gimeno, I. Von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang *et al.*, "Solving mixed integer programs using neural networks," *arXiv preprint arXiv:2012.13349*, 2020.

[15] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html

[16] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[17] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] V. Garg, S. Jegelka, and T. Jaakkola, "Generalization and representational limits of graph neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 3419–3430.

[20] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[21] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *CoRR*, vol. abs/2003.00982, 2020. [Online]. Available: https://arxiv.org/abs/2003.00982

[22] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of the web conference 2020*, 2020, pp. 2704–2710.

[23] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[24] H. Hankimaa, "Optimising energy and reserve offers in the nordic markets under uncertainty," 2023, unpublished MSc thesis. [Online]. Available: https://urn.fi/URN:NBN:fi:aalto-202308275171

[25] S. Zhu, C. Zhou, S. Pan, X. Zhu, and B. Wang, "Relation structure-aware heterogeneous graph neural network," in *2019 IEEE international conference on data mining (ICDM)*. IEEE, 2019, pp. 1534–1539.

[26] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf

[27] F. Zhang, X. Liu, J. Tang, Y. Dong, P. Yao, J. Zhang, X. Gu, Y. Wang, B. Shao, R. Li, and K. Wang, "Oag: Toward linking large-scale heterogeneous entity graphs," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2585–2595. [Online]. Available: https://doi-org.libproxy.aalto.fi/10.1145/3292500.3330785

[28] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," *arXiv preprint arXiv:2006.05205*, 2020.

[29] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform bad for graph representation?" 2021.

[30] M. S. Hussain, M. J. Zaki, and D. Subramanian, "Global self-attention as a replacement for graph convolution," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 655–665.

[31] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

A Hyperparameter tuning

The model training and hyperparameter tuning was performed using up to 10 Nvidia V100 GPUs in parallel. The hyperparameter search domains are listed in Table A1 with the best hyperparameters highlighted. We used a batch size of 64 problem instances.

Table A1: Hyperparameter search domains with the chosen parameters in bold.

Hyperparameter	Search domain
γ	{0.0001, 0.0003, 0.001, **0.003**}
n_{head}	{2, 4, **8**, 16}
d	{**16**, 24, 36, 54, 81, 122}
T	{2, 3, 6, 10, **15**, 23, 34, 50, 75}