Noname manuscript No.
(will be inserted by the editor)

Learning Heuristics for Transit Network Design and Improvement with Deep Reinforcement Learning

Andrew Holliday · Ahmed El-Geneidy · Gregory Dudek

the date of receipt and acceptance should be inserted later

Abstract Transit agencies world-wide face tightening budgets and declining ridership. To maintain quality of service while cutting costs, efficient transit network design is essential. But planning a network of public transit routes is a challenging optimization problem. The most successful approaches to date use metaheuristic algorithms to search through the space of possible transit networks by applying low-level heuristics that randomly alter routes in a network. The design of these low-level heuristics has a major impact on the quality of the result. In this paper we use deep reinforcement learning with graph neural nets to learn low-level heuristics for an evolutionary algorithm, instead of designing them manually. These learned heuristics improve the algorithm's results on benchmark synthetic cities with 70 nodes or more, and achieve new state-of-the-art results the challenging Mumford benchmark. They also improve upon a simulation of the real transit network in the city of Laval, Canada, by as much as 52% and 25% on two key metrics, and offer cost savings of up to 19% over the city's existing transit network.

Acknowledgements

We thank the Société de Transport de Laval for the map data and transportation data they provided, and the Agence Métropolitaine de Transport of Quebec for the origin-destination dataset they provided. This data was essential to the experiments presented in this paper. We also gratefully acknowledge the funding we received from NSERC in support of this work. And we thank Joshua Katz for acting as a sounding board for ideas and giving feedback on our experimental designs.

Address(es) of author(s) should be given

1 INTRODUCTION

The COVID-19 pandemic resulted in declines in transit ridership in cities worldwide (Liu et al., 2020). This has led to a financial crisis for municipal transit agencies, putting them under pressure to reduce transit operating costs (Kar et al., 2022). But if cost-cutting leads to reduced quality of service, ridership may be harmed further, leading to a downward spiral of service quality and income. Agencies find themselves having to do more with less. This makes the spatial design of a transit network very important, as a well-planned layout can cut operating costs and improve service quality. But changes to a transit network are not to be made lightly. Network redesigns can be costly and disruptive for users and bus drivers, and for rail transit, it can require infrastructural changes that may be prohibitive. So it is vital to agencies that they get network design or network changes right the first time. Good algorithms for the transit network design problem (NDP) can therefore be very useful to transit agencies.

The NDP is the problem of designing a set of transit routes for a city that satisfy one or more objectives, such as meeting all travel demand and minimizing operating costs. It is an NP-hard problem, and since real-world cities typically have hundreds or even thousands of possible stop locations, analytical optimization approaches are usually infeasible. The most successful approaches to the NDP to-date have been metaheuristic algorithms. Most such algorithms work by repeatedly applying one or more "low-level" heuristics that randomly modify a network, guiding this random walk towards better networks over many iterations by means of a metaheuristic such as natural selection (as in evolutionary algorithms) or metallic annealing (as in simulated annealing). Many different metaheuristics for guiding the search have been proposed, as have a range of low-level heuristics for transit network design specifically. While these algorithms can find good solutions in many cases, real-world transit networks are still commonly designed by hand (Durán-Micco and Vansteenwegen, 2022).

The different low-level heuristics used in these algorithms make different kinds of changes to a network, such as randomly removing a stop on an route, or randomly exchanging two stops on a route. What most have in common is that they are uniformly random: no aspect of the particular network or city affects the heuristic's likelihood of making one change versus another. In this work, we investigate whether deep reinforcement learning (DRL) can be used to learn more intelligent heuristics that use information about the city and the current transit network to make only the most promising changes. We find that indeed, heuristics learned in this way can find better transit networks than state-of-the-art algorithms that use only random heuristics.

In our prior work (Holliday and Dudek, 2023), we trained a graph neural net (GNN) to plan a transit network from scratch, and showed that using this to generate an initial transit network improved the quality of networks found by two metaheuristic algorithms. In subsequent work (Holliday and Dudek, 2024) we investigated whether the GNN can benefit a metaheuristic algorithm by serving as a learned low-level heuristic.

In the present work, we expand on those results in several ways. We present new results obtained with an improved GNN model trained via Proximal Policy Optimization (PPO) instead of the older REINFORCE algorithm, and use an improved evolutionary algorithm. We evaluate our algorithm on the the widely-used Mandl (Mandl, 1980) and Mumford (Mumford, 2013a) benchmark cities, and find that on the most challenging of these cities, our algorithm surpasses the previous state of the art, improving on prior results by up to 4.8%.

We also perform ablation studies to analyze the importance of different components of our neural-evolutionary hybrid algorithm. As part of these ablations, we consider a novel unlearned low-level heuristic, where shortest paths with common end-points are joined uniformly at random instead of according to the GNN's output. Interestingly, we find that in the narrow case where we are only concerned with minimizing passenger travel time, this heuristic outperforms both the baseline heuristics of Nikolić and Teodorović (2013) and our GNN heuristic; in most other cases, however, our GNN heuristic performs better.

Finally, we go beyond our earlier work by applying our heuristics to real data from the city of Laval, Canada, a very large real-world problem instance. We show that our learned heuristic can be used to plan transit networks for Laval that, in simulation, exceed the performance of the city's existing transit system by a wide margin, for three distinct optimization goals. These results show that learned GNN heuristics may allow transit agencies to offer better service at less cost.

2 BACKGROUND AND RELATED WORK

2.1 Deep Learning for Optimization Problems

Deep learning refers to machine learning techniques that involve "deep" artificial neural nets - that is, neural nets with many hidden layers. Deep reinforcement learning refers to the use of deep neural nets in reinforcement learning (RL). RL is a branch of machine learning in which a learning system, such as a neural net, chooses actions and receives a numerical "reward" for each action, and learns to choose actions that maximize total reward. As documented by Bengio et al. (2021), there is growing interest in the application of deep learning, and deep RL specifically, to combinatorial optimization (CO) problems.

Vinyals et al. (2015) proposed a deep neural net model called a Pointer Network, and trained it via supervised learning to solve instances of the travelling salesman problem (TSP). Subsequent work, such as that of Dai et al. (2017), Kool et al. (2019), and Sykora et al. (2020), has built on this. These works use similar neural net models, in combination with reinforcement learning algorithms, to train neural nets to construct CO solutions. These have attained impressive performance on the TSP, the vehicle routing problem (VRP), and other CO problems. More recently, Mundhenk et al. (2021) train an recurrent neural net (RNN) via RL to construct a starting population of solutions to a genetic algorithm, the outputs of which are used to further train the RNN. Fu et al. (2021) train a model on small TSP instances and present an algorithm that applies the model to much larger instances. Choo et al. (2022) present a hybrid algorithm of Monte Carlo Tree Search and Beam Search that draws better sample solutions for the TSP and Capacitated VRP (CVRP) from a neural net policy like that of Kool et al. (2019).

These approaches all belong to the family of "construction" methods, which solve a CO problem by starting with an "empty" solution and adding to it until it is complete - for example, in the TSP, this would mean constructing a path one node at a time, starting with an empty path and stopping once the path includes all nodes. The solutions from these neural construction methods come close to the quality of those from specialized algorithms such as Concorde (Applegate et al., 2001), while requiring much less run-time to compute (Kool et al., 2019).

By contrast with construction methods, “improvement” methods start with a complete solution and repeatedly modify it, searching through the solution space for improvements. In the TSP example, this might involve starting with a complete path, and swapping pairs of nodes in the path at each step to see if the path is shortened. Improvement methods are generally more computationally costly than construction methods but can yield better results. Evolutionary algorithms belong to this category.

Some work (Hottung and Tierney, 2019; Chen and Tian, 2019; da Costa et al., 2020; Wu et al., 2021; Ma et al., 2021), has considered training neural nets to choose the search moves to be made at each step of an improvement method. meanwhile Kim et al. (2021) train one neural net to construct a set of initial solutions, and another to modify and improve them. And more recently, Ye et al. (2024) train a neural net to provide a heuristic score for choices in CO problems in the context of an ant colony optimization algorithm. This work has shown impressive performance on the TSP, VRP, and similar CO problems.

In most of the above work, the neural net models used are graph neural nets, a type of neural net model that is designed to operate on graph-structured data (Bruna et al., 2013; Kipf and Welling, 2016; Defferrard et al., 2016; Duvenaud et al., 2015). These have been applied in many other domains, including the analysis of large web graphs (Ying et al., 2018), the design of printed circuit boards (Mirhoseini et al., 2021), and the prediction of chemical properties of molecules (Duvenaud et al., 2015; Gilmer et al., 2017). An overview of GNNs is provided by Battaglia et al. (2018). Like many CO problems, the NDP lends itself to being described as a graph problem, so we use GNN models here as well.

CO problems also have in common that it is difficult to find a globally optimal solution but easier to gauge the quality of a given solution. As Bengio et al. (2021) note, this makes reinforcement learning a natural fit to CO problems. Most of the work cited in this section uses RL methods to train neural net models.

2.2 Optimization of Public Transit

The above work all concerns a small set of classical CO problems including the TSP and VRP. Like the NDP, some of these are NP-hard, and all can be described as graph problems. But the NDP resembles these classical problems much less than they do each other: transit routes can interact by transferring passengers, and the space of solutions for a problem instance of given size is much vaster in the NDP. Take as an example the small Mumford0 benchmark city described in Table 1. With 30 nodes there are $30! \approx 2.7 \times 10^{32}$ possible TSP tours, but for 12 routes with at least 2 and at most 15 stops each, there are approximately 2.3×10^{11} possible routes and so $\binom{2.3\times10^{11}}{12} \approx 4.3 \times 10^{127}$ possible transit networks, a factor of 10^{95} more.

The difference is further shown by the fact that the state-of-the-art on classic problems like the TSP uses analytical and mathematical programming methods (such as Applegate et al. (2001)), but on the NDP such methods are not used. The NDP's differences necessitate a special treatment, and so all recent published work on the NDP deals with the NDP alone (Mumford, 2013a; John et al., 2014; Kılıç and Gök, 2014; Islam et al., 2019; Ahmed et al., 2019; Lin and Tang, 2022; Hüsselmann et al., 2023), and we do the same here.

While analytical and mathematical programming methods have been successful on small instances (van Nes, 2003; Guan et al., 2006), they struggle to realistically represent the problem (Guihaire and Hao, 2008; Kepaptsoglou and Karlaftis, 2009). Metaheuristic approaches, as defined by Sörensen et al. (2018), have thus been more widely applied.

The most widely-used metaheuristics for the NDP have been genetic algorithms, simulated annealing, and ant-colony optimization, along with hybrids of these methods (Guihaire and Hao, 2008; Kepaptsoglou and Karlaftis, 2009; Durán-Micco and Vansteenwegen, 2022; Yang and Jiang, 2020; Hüsselmann et al., 2023). Recent work has also shown other metaheuristics can be used with success, such as sequence-based selection hyperheuristics (Ahmed et al., 2019), beam search (Islam et al., 2019), and particle swarms (Lin and Tang, 2022). Many different low-level heuristics have been applied within these metaheuristic algorithms, but most have in common that they select among possible neighbourhood moves uniformly at random.

One exception is Hüsselmann et al. (2023). For two heuristics, the authors design a simple model of how each change the heuristic could make would affect solution quality. They use this model to give the different changes different probabilities of being selected. The resulting heuristics obtain state-of-the-art results. However, their simple model ignores passenger trips involving transfers, and the impact of the user's preferences over different parts of the cost function. By contrast, the method we propose **learns** a model of changes' impacts to assign probabilities to those changes, and does so based on a richer set of inputs.

While neural nets have often been used for predictive problems in urban mobility (Xiong and Schneider, 1992; Rodrigue, 1997; Chien et al., 2002; Jeong and Rilett, 2004; Çodur and Tortum, 2009; Li et al., 2020) and for other transit optimization problems such as scheduling, passenger flow control, and traffic signal control (Zou et al., 2006; Ai et al., 2022; Yan et al., 2023; Jiang et al., 2018; Wang et al., 2024), they have not often been applied to the NDP, and neither has RL. Two recent examples are Darwish et al. (2020) and Yoo et al. (2023). Both use RL to design routes and a schedule for the Mandl benchmark (Mandl, 1980), a single small city with just 15 transit stops, and both obtain good results. Darwish et al. (2020) use a GNN approach inspired by Kool et al. (2019); in our own work we experimented with a nearly identical approach to Darwish et al. (2020), but found it did not scale beyond very small instances. Meanwhile, Yoo et al. (2023) use tabular RL, a type of approach which is practical only for small problem sizes. Both of these approaches also require a new model to be trained on each problem instance. Our approach, by contrast, is able to find good solutions for NDP instances of more than 600 nodes, and can be applied to instances unseen during model training.

3 TRANSIT NETWORK DESIGN PROBLEM

In the transit network design problem, one is given an augmented graph that represents a city:

$$\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D) \tag{1}$$

This is comprised of a set $\mathcal{N}$ of n nodes, representing candidate stop locations; a set $\mathcal{E}_s$ of street edges (i, j, τ_{ij}) connecting the nodes, with weights τ_{ij} indicating drive times on those streets; and an $n \times n$ Origin-Destination (OD) matrix D giving the travel demand, in number of trips, between every pair of nodes in $\mathcal{N}$. The goal is to propose a transit network, which is a set of routes $\mathcal{R}$ that minimizes a cost function $C : \mathcal{C}, \mathcal{R} \rightarrow \mathbb{R}^+$, where each route $r \in \mathcal{R}$ is a sequence of nodes in $\mathcal{N}$. The transit network $\mathcal{R}$ is subject to the following constraints:

1. $\mathcal{R}$ must satisfy all demand, providing some path over transit between every pair of nodes $(i, j) \in \mathcal{N}$ for which $D_{ij} > 0$.
2. $\mathcal{R}$ must contain exactly S routes ($|\mathcal{R}| = S$), where S is a parameter set by the user.
3. All routes $r \in \mathcal{R}$ must obey the same limits on its number of stops: $MIN \leq |r| \leq MAX$, where MIN and MAX are parameters set by the user.

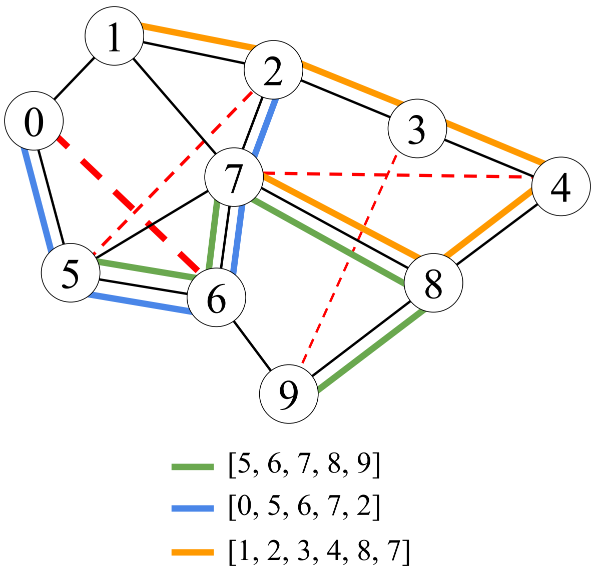


Fig. 1: An example city graph with ten numbered nodes and three routes. Street edges are black, routes are in colour, and demands are shown by dashed red lines. The edges of the three routes form a subgraph of the street graph $(\mathcal{N}, \mathcal{E}_s)$. All node-pairs with demand are connected by this subgraph, so the three routes form a valid transit network. The demand between nodes 2 and 5 and between 0 and 6 can be satisfied directly by riding on the blue line, and the demand from 7 to 4 by the orange line. But the demand from 3 to 9 requires one transfer: passengers must ride the orange line from node 3 to 8, and then the green line from node 8 to 9.

4. Routes $r \in \mathcal{R}$ must not contain cycles; each node $i \in \mathcal{N}$ can appear in r at most once.

We here deal with the symmetric NDP, meaning that all demand, streets, and routes are the same in both directions. This means $D = D^\top$, $(i, j, \tau_{ij}) \in \mathcal{E}_s$ iff. $(j, i, \tau_{ij}) \in \mathcal{E}_s$, and all routes are traversed both forwards and backwards by vehicles on them. An example city graph with a transit network is shown in Figure 1.

3.1 Cost Function

We take the NDP cost function to have three components. The cost to passengers, C_p, is the average passenger trip time over the network:

$$C_p(\mathcal{C}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij}(\delta_{\mathcal{R}ij}\tau_{\mathcal{R}ij} + (1 - \delta_{\mathcal{R}ij})2\max_{kl} T_{kl})}{\sum_{i,j} D_{ij}} \tag{2}$$

Where $\tau_{\mathcal{R}ij}$ is the time of the shortest transit trip from i to j given $\mathcal{R}$, including a time penalty p_T for each transfer the rider must make between routes, and $\delta_{\mathcal{R}ij}$ is a delta function that takes value 1 if $\mathcal{R}$ provides a path from i to j, and 0 if not. If $\mathcal{R}$ satisfies constraint 1, it connects all pairs of nodes, so for all feasible networks the above equation simplifies to:

$$C_p(\mathcal{C}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij}\tau_{\mathcal{R}ij}}{\sum_{i,j} D_{ij}} \tag{3}$$

Otherwise, each demand that is unsatisfied is treated as taking twice the largest inter-node driving distance to be satisfied.

The operating cost is the total driving time of the routes, or total route time:

$$C_o(\mathcal{C}, \mathcal{R}) = \sum_{r\in\mathcal{R}} \tau_r \tag{4}$$

Where τ_r is the time needed to completely traverse a route r in one direction.

To enforce the constraints on $\mathcal{R}$, we use a third term C_c, itself the sum of three terms:

$$C_c(\mathcal{C}, \mathcal{R}) = F_{un} + F_s + 0.1\delta_v \tag{5}$$

Here, F_{un} is the fraction of node-pairs (i, j) with $D_{ij} > 0$ for which $\mathcal{R}$ provides no path: a measure of how many possible violations of constraint 1 really occur in $\mathcal{R}$. F_s is a similar measure for constraint 3, but it is proportional to the actual number of stops more than MAX or less than MIN that each route has. Specifically:

$$F_s = \frac{\sum_{r\in\mathcal{R}} \max(0, MIN - |r|, |r| - MAX)}{S * MAX} \tag{6}$$

δ_v is a delta function that takes value 0 if $F_{un} = 0$ and $F_s = 0$, and takes value 1 otherwise.

We use fractional measures F_{un} and F_s instead of absolute measures so that C_c will tend to fall in the range $[0, 1]$ regardless of the size of the city graph. This is desirable for reasons relating to the numerical stability of neural net training. However, it has the drawback that if the city graph C has very many nodes (large n), and if $\mathcal{R}$ violates only a few constraints, F_{un} and F_s may become vanishingly small compared to C_o and C_p. This could lead to algorithms ignoring small numbers of constraint violations. To prevent this, we include the term $0.1\delta_v$, which ensures that if any constraints are violated, C_c cannot be less than 0.1, but it will be 0 if all constraints are respected. This ensures that even one violation will have a significant impact on overall cost $C(\mathcal{C}, \mathcal{R})$, no matter the size of the city graph.

C_c does not penalize violations of constraints 2 and 4, because these constraints are enforced directly by the design of the Markov Decision Process (MDP) detailed in subsection 3.2.

The cost function is then:

$$C(\mathcal{C}, \mathcal{R}) = \alpha w_p C_p + (1 - \alpha) w_o C_o + \beta C_c \tag{7}$$

The weight $\alpha \in [0, 1]$ controls the trade-off between passenger and operating costs, while β is the penalty assigned for each constraint violation. w_p and w_o are rescaling constants chosen so that $w_p C_p$ and $w_o C_o$ both vary roughly over the range $[0, 1)$ for different $\mathcal{C}$ and $\mathcal{R}$; this is done so that α will properly balance the two, and to stabilize training of the GNN policy. The values used are $w_p = (\max_{i,j} T_{ij})^{-1}$ and $w_o = (S \max_{i,j} T_{ij})^{-1}$, where T is an $n \times n$ matrix of shortest-path driving times between every node pair.

3.2 Markov Decision Process Formulation

A Markov Decision Process (MDP) is a formalism commonly used to define problems in RL (Sutton and Barto, 2018, Chapter 3). In an MDP, an **agent** interacts with an **environment** over a series of **timesteps** t, starting at $t = 0$. At each timestep t, the environment is in a **state** $s_t \in \mathcal{S}$, and the agent observes the state and takes some **action** $a_t \in \mathcal{A}_t$, where $\mathcal{A}_t$ is the set of available actions at that timestep. The environment then transitions to a new state $s_{t+1} \in \mathcal{S}$ according to the state transition distribution $P(s_{t+1}|s_t, a_t)$, and the agent receives a numerical **reward** $R_t \in \mathbb{R}$ according to the reward distribution $P(R_t|s_t, a_t, s_{t+1})$. The agent chooses actions according to its **policy** $\pi(a_t|s_t)$, which is a probability distribution over $\mathcal{A}_t$ given s_t. In RL, the goal is usually to learn a policy π that maximizes the return G_t, defined as a time-discounted sum of rewards:

$$G_t = \sum_{t'=t}^{t_{\text{end}}} \gamma^{t'-t} R_{t'} \tag{8}$$

Where $\gamma \in [0, 1]$ is a parameter that discounts rewards farther in the future, and t_{end} is the final timestep of the MDP. The sequence of states visited, actions taken, and rewards received from $t = 0$ to t_{end} constitutes one **episode** of the MDP.

We here describe the MDP we use to represent a construction approach to the transit network design problem. As shown in Equation 9, the state s_t is composed of the set of finished routes $\mathcal{R}_t$, and an in-progress route r_t which is currently being planned.

$$s_t = (\mathcal{R}_t, r_t) \tag{9}$$

The starting state is $s_1 = (\mathcal{R}_1 = \{\}, r_1 = [])$. At a high level, the MDP alternates at every timestep between two modes: on odd-numbered timesteps, the agent selects an extension to the route r_t that it is currently planning; on even-numbered timesteps, the agent chooses whether or not to stop extending r_t and add it to the set of finished routes.

On odd-numbered timesteps, the available actions are drawn from SP, the set of shortest paths between all pairs of nodes in $\mathcal{C}$. If $r_t = []$, then:

$$\mathcal{A}_t = \{a \mid a \in \text{SP}, |a| \leq MAX\} \tag{10}$$

Otherwise, $\mathcal{A}_t = \text{EX}_{r_t}$, where EX_{r_t} is the set of paths $a \in \text{SP}$ that satisfy all of the following conditions:

- $(i, j, \tau_{ij}) \in \mathcal{E}_s$, where i is the first node of a and j is the last node of r_t, or vice-versa
- a and r_t have no nodes in common
- $|a| \leq MAX - |r_t|$

Once a path $a_t \in \mathcal{A}_t$ is chosen, r_{t+1} is formed by appending a_t to the beginning or end of r_t as appropriate.

On even-numbered timesteps, the action space depends on the number of stops in r_t:

$$\mathcal{A}_t = \begin{cases} \{\text{continue}\} & \text{if} |r_t| < MIN \\ \{\text{halt}\} & \text{if} |r_t| = MAX \text{ or } |\text{EX}_{r_t}| = 0 \\ \{\text{continue}, \text{halt}\} & \text{otherwise} \end{cases} \tag{11}$$

If $a_t = \text{halt}$, r_t is added to $\mathcal{R}_t$ to get $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_t\}$, and $r_{t+1} = []$ is a new empty route. If $a_t = \text{continue}$, then $\mathcal{R}_{t+1} = \mathcal{R}_t$ and $r_{t+1} = r_t$.

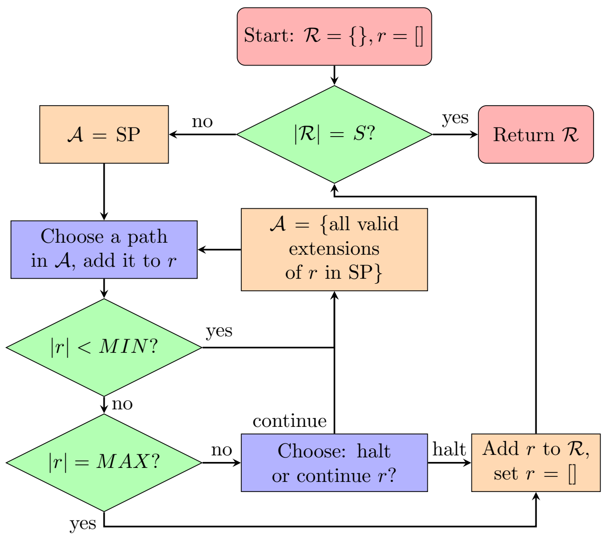
The episode ends when the S-th route is added to $\mathcal{R}$: that is, if $|\mathcal{R}_{t+1}| = S$, the episode ends at timestep t. The output transit network is set to $\mathcal{R} = \mathcal{R}_{t+1}$. The reward function is defined as the decrease in cost between each consecutive pair of timesteps:

$$R_t = C'(\mathcal{C}, \mathcal{R}_t \cup \{r_t\}) - C'(\mathcal{C}, \mathcal{R}_{t+1} \cup \{r_{t+1}\}) \tag{12}$$

Note that this means that on even timesteps (when halt-or-continue actions are taken), $R_t = 0$ because $\mathcal{R}_t$ and r_t do not change as a result of a halt-or-continue action.

In the above equation, $C'((\mathcal{C}, \mathcal{R}) = C(\mathcal{C}, \mathcal{R}) - \beta F_s$; that is, the cost function without the F_s component that penalizes violations of constraint 3. We remove this component because, by construction of the MDP, the networks it produces always satisfy constraint 3, so there is no need to penalize violations of it that may occur before it is complete.

This MDP formalization imposes some helpful biases on the space of solutions. First, it requires any route connecting i and j to stop at all nodes along some path between i and j, biasing planned routes towards covering more nodes. Second, it biases routes towards



Start: $\mathcal{R} = \{\}, r = []$
$|\mathcal{R}| = S$?
no
yes
$\mathcal{A}$ = SP
Return $\mathcal{R}$
Choose a path in $\mathcal{A}$, add it to r
$\mathcal{A}$ = {all valid extensions of r in SP}
$|r| < MIN$?
yes
no
continue
$|r| = MAX$?
no
Choose: halt or continue r?
halt
Add r to $\mathcal{R}$, set r = []
yes

Fig. 2: A flowchart of the transit network construction process defined by our MDP. Blue boxes indicate points where the timestep t is incremented and the agent selects an action. Red nodes are the beginning and ending of the process. Green nodes are hard-coded decision points. Orange nodes show updates to the state $\mathcal{S}$ and action space $\mathcal{A}$.

directness by forcing them to be composed of shortest paths. While an agent may construct arbitrarily indirect routes by choosing paths with length 2 at every step, this is unlikely because in a realistic street graph, the majority of paths in SP are longer than two nodes. Third, the alternation between deciding to continue or halt a route and deciding how to extend the route means that the probability of halting does not depend on how many different extensions are available; so a policy learned in environments with fewer extensions should generalize more easily to environments with more, and vice versa.

4 NEURAL NET HEURISTICS

4.1 Learning to Construct a Network

In order to learn heuristics for transit network design, we first train a GNN policy $\pi_\theta(a|s)$, parameterized by θ, to maximize the cumulative return G_t on the construction MDP described in subsection 3.2. By then following this learned policy on the MDP for some city $\mathcal{C}$, stochastically sampling a_t at each step with probabilities given by $\pi(\cdot|s_t)$, we can obtain a transit network $\mathcal{R}$ for that city. We refer to this construction algorithm using the learned policy as "learned construction". Because learned construction samples actions stochastically, we can run it multiple times to generate multiple networks for a city, and then pick the lowest-cost one. We will often do this by running learned construction 100 times and choosing the best network; we denote this procedure "LC-100". As noted in subsection 2.2, the NDP has a much vaster space of possible solutions than problems like the TSP or VRP. To make this vast problem tractable, we found it necessary to "factorize" the computation of heuristic scores for each shortest-path extension to a route. Our policy net thus outputs a scalar score o_{ij} for each node pair (i, j) in the graph at each step. From these, a score for each extension o_a is computed by summing over o_{ij} for all (i, j) that would become directly connected by appending path a to the current route r_t:

$$o_a = \sum_{i \in r_t} \sum_{j \in a} o_{ij} + \sum_{k \in a} \sum_{l \in a} o_{kl}$$

Where $o_{ii} = 0$ for all i. This factorization makes the learning objective "estimate the benefit of directly connecting nodes i and j with a given travel time", rather than the much more complex "estimate the quality of adding path a to r_t". This objective is easier to learn, and reduces the needed complexity of the neural net architecture.

The central component of the policy net is a graph attention net (GAT) (Brody et al., 2021) which treats the city as a fully-connected graph on the nodes $\mathcal{N}$. Each node has an associated feature vector $\mathbf{x}_i$, and each edge a feature vector $\mathbf{e}_{ij}$, containing information about location, demand, existing transit connections, and the street edge (if one exists) between i and j. We note that a graph attention net operating on a fully-connected graph has close parallels to a Transformer model (Vaswani et al., 2017), but unlike a Transformer, this architecture enables the use of edge features that describe known relationships between elements.

The GAT outputs node embeddings $\mathbf{y}_i$, which are operated on by one of two "head" neural nets, depending on the timestep: NN_{ext} and NN_{halt}. NN_{ext} computes the o_{ij} node-pair scores from the node embeddings $\mathbf{y}_i$ and the driving time from i to j along $r_t|a$ when the timestep t is odd. NN_{halt} decides whether to halt when t is even, based on $\mathbf{y}_i$ and on other statistics of the city $\mathcal{C}$ and partial network $\mathcal{R}_t, r_t$. The full architectural details of these components are presented in Appendix A, along with the details of the input node and edge features.

We have released the code for our method and experiments to the public[2].

[2] Available at `https://github.com/AHolliday/transit_learning`

4.1.1 Training

In our prior work (Holliday and Dudek, 2023, 2024), we trained the policy net using the REINFORCE with Baseline method proposed by Williams (1992), using a reward function that was 0 at all steps except the final step, where it was the negative of the cost function. This was inspired by the similar approach taken by Kool et al. (2019). However, REINFORCE has been been built upon by a number of other policy gradient learning algorithms since its publication. Also, sparse reward functions like this one, which is zero at most timesteps, convey little information about the effect of each action, and so can make learning difficult.

For these reasons, in this work we train the policy net using PPO, a policy gradient method proposed by Schulman et al. (2017), and use the richer reward function described in section 3.2. The PPO method was chosen for its relative simplicity and for the fact that it remains a state-of-the-art method on many problems. PPO works by rolling out a number of steps H of a "batch" of episodes in parallel, and then computing an "advantage" for each timestep, $A_t = \left(\sum_{t'=0}^{H-1} \gamma^{t'} R_{t+t'}\right) + \gamma^H V(s_{t+H}) - V(s_t)$. This advantage weights the updates to the policy, such that actions with a positive advantage are made more likely, and those with a negative advantage are made less so. In this work, we use Generalized Advantage Estimation as described by Schulman et al. (2017): we use a generalized advantage, $\hat{A}_t$, that is a weighted sum of A_t over a range of horizon values H.

The value function $V(s_t)$ is itself a small Multi-Layer Perceptron (MLP) neural net that is trained in parallel with the policy to predict the discounted return $G_t^H = \sum_{t'=0}^{H-1} \gamma^{t'} R_{t+t'} + \gamma^H V(s_{t+H})$, based on the cost weight parameter α and statistics of the city $\mathcal{C}$ and of the current network $\mathcal{R}_t, r_t$.

We train the policy net on a dataset of synthetic cities. We sample a different $\alpha \sim [0, 1]$ for each city, while holding S, MIN, MAX, and the constraint weight β constant across training. The values of these parameters used are presented in Table 7. At each iteration, a full MDP episode is run on a "batch" of cities from the dataset, $C(\mathcal{C}, \mathcal{R})$, $V(s_t)$ and G_t^H are computed across the batch, and back-propagation and parameter updates are applied to both the policy net and the value net according to PPO.

To construct a synthetic city for the training dataset, we first generate its nodes and street network using one of these processes chosen at random:

– 4-nn: Sample n random 2D points uniformly in a square to give $\mathcal{N}$. Add street edges to each node i from its four nearest neighbours.

– 4-grid: Place n nodes in a rectangular grid as close to square as possible. Add edges between all horizontal and vertical neighbour nodes.
– 8-grid: The same as 4-grid, but also add edges between diagonal neighbour nodes.
– Voronoi: Sample m random 2D points in a square, and compute their Voronoi diagram (Fortune, 1995). Take the shared vertices and edges of the resulting Voronoi cells as $\mathcal{N}$ and $\mathcal{E}_s$. m is chosen so $|\mathcal{N}| = n$.

For each process except Voronoi, each edge in $\mathcal{E}_s$ is then deleted with probability ρ. If the resulting street graph is not strongly connected, it is discarded and the process is repeated.

To generate our training dataset, we set $n = 20$ and $\rho = 0.3$, sample nodes (or the m points for Voronoi) in a 30km × 30km square, and assume a fixed vehicle speed of $v = 15$m/s to compute street edge weights $\tau_{ij} = ||(x_i, y_i) - (x_j, y_j)||_2/v$. Finally, we generate the OD matrix D by setting diagonal demands $D_{ii} = 0$, uniformly sampling above-diagonal elements D_{ij} in the range $[60, 800]$, and then setting below-diagonal elements $D_{ji} = D_{ij}$ to make D symmetric. The dataset is composed of $2^{15} = 32,768$ synthetic cities generated in this way.

During training, we make a 90:10 split of this dataset into training and validation sets. In each iteration of training, we run learned construction with the policy net on one MDP episode over each city in a batch that is sampled from the dataset, and update the policy and value nets. After every ten iterations, we run learned construction with the policy net on each city in the validation set, and the average cost of the resulting networks is recorded. At the end of training, the parameters θ that achieved the lowest average validation cost are returned, giving the final policy π_θ. Training proceeds for 200 iterations in batches of 256 cities, as we found that the policy stopped improving after this. A total of 51,200 cities are used during training: less than the entire training dataset.

All neural net inputs are normalized so as to have unit variance and zero mean across the entire dataset during training. The scaling and shifting normalization parameters are saved as part of the policy net and are applied to new data presented to π_θ after training.

4.2 Evolutionary Algorithm

We wish to consider whether our learned policy π_θ can be effective as a low-level heuristic in a metaheuristic algorithm. We test this by comparing a simple evolutionary algorithm to a variant of the algorithm where we replace one of its low-level heuristics with one that uses our learned policy. In this section we describe the baseline evolutionary algorithm and our variant of it.

Our baseline evolutionary algorithm is based on that of Nikolić and Teodorović (2013), with several modifications that we found improved its performance. We chose this algorithm because its ease of implementation and its good performance at short running times (with appropriate parameters), aided prototyping and experimentation, but we note that our learned policies can in principle be used as heuristics in a wide variety of metaheuristic algorithms.

The algorithm operates on a population of B solutions $\mathcal{B} = \{\mathcal{R}_b | 1 \leq b \leq B\}$, and performs alternating stages of mutation and selection. In the mutation stage, the algorithm applies two "mutators" (ie. low-level heuristics), type 1 and type 2, to equally-sized subsets of the population chosen at random; if the mutated network $\mathcal{R}'_b$ has lower cost than its "parent" $\mathcal{R}_b$, it replaces its parent in $\mathcal{B}$: $\mathcal{R}_b \leftarrow \mathcal{R}'_b$. This is repeated E times in the stage. Then, in the selection stage, solutions either "die" or "reproduce", with probabilities inversely related to their cost $C(\mathcal{C}, \mathcal{R}_b)$. After IT repetitions of mutation and selection, the algorithm returns the best network found over all iterations.

Both mutators begins by selecting, uniformly at random, a route r in $\mathcal{R}_b$ and a terminal node i on that route. The type-1 mutator then selects a random node $j \neq i$ in $\mathcal{N}$, and replaces r with the shortest path between i and j, SP_{ij}. The probability of choosing each node j is proportional to the amount of demand directly satisfied by SP_{ij}. The type-2 mutator chooses with probability p_d to delete i from r; otherwise, it adds a random node j in i's street-graph neighbourhood to r (before i if i is the first node in r, and after i if i is the last node in r), making j the new terminal. Following Nikolić and Teodorović (2013), we set the deletion probability $p_d = 0.2$ in our experiments.

The initial population of solutions is constructed by making B copies of a single initial solution, $\mathcal{R}_0$. In our prior work (Holliday and Dudek, 2023), we found that using learned construction to plan $\mathcal{R}_0$ outperformed other methods, so we use this technique here. Specifically, we use LC-100 to generate $\mathcal{R}_0$.

The parameters of this algorithm are the population size B, the number of mutations per mutation stage E, and the number of iterations of mutation and selection stages IT. In addition to these, the algorithm takes as input the city $\mathcal{C}$ being planned over, the set of shortest paths through the city's street graph SP, and the cost function C. The full procedure is given in Algorithm 1. We refer to this algorithm as "EA".

Algorithm 1 Evolutionary Algorithm

```
1: Input: $\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D), \mathrm{SP}, C, B, IT, E$
2: Construct initial network $\mathcal{R}_0 \leftarrow$ LC-100$(\mathcal{C}, C)$
3: $\mathcal{R}_b \leftarrow \mathcal{R}_0 \;\forall\; b \in$ integers 1 through $B$
4: $\mathcal{R}_{\text{best}} \leftarrow \mathcal{R}_0$
5: for $i = 1$ to $IT$ do
6:     // Mutation stage
7:     for $j = 1$ to $E$ do
8:         for $b = 1$ to $B$ do
9:             if $b \leq B/2$ then
10:                $\mathcal{R}'_b \leftarrow$ type_1_mut$(\mathcal{R}_b)$
11:            else
12:                $\mathcal{R}'_b \leftarrow$ type_2_mut$(\mathcal{R}_b)$
13:            if $C(\mathcal{C}, \mathcal{R}'_b) < C(\mathcal{C}, \mathcal{R}_b)$ then
14:                $\mathcal{R}_b \leftarrow \mathcal{R}'_b$
15:        Randomly shuffle network indices $b$
16:    // Selection stage
17:    $C_{max} \leftarrow \max_b C(\mathcal{C}, \mathcal{R}_b)$
18:    $C_{min} \leftarrow \min_b C(\mathcal{C}, \mathcal{R}_b)$
19:    for $b = 1$ to $B$ do
20:        if $C(\mathcal{C}, \mathcal{R}_b) < C(\mathcal{C}, \mathcal{R}_{\text{best}})$ then
21:            $\mathcal{R}_{\text{best}} \leftarrow \mathcal{R}_b$
22:        // Select surviving networks
23:        $O_b \leftarrow \frac{C_{max} - C_b}{C_{max} - C_{min}}$
24:        $s_b \sim$ Bernoulli$(1 - e^{-O_b})$
25:        // Set survivor reproduction probabilities
26:        $p_b \leftarrow \frac{O_b s_b}{\sum_{b'} O_{b'} s_{b'}}$
27:    if $\exists\; b \in [1, B]$ s.t. $s_b = 1$ then
28:        // Replace non-survivors with survivors' offspring
29:        for $b = 1$ to $B$ do
30:            if $s_b = 0$ then
31:                $k \sim P(K)$, where $P(K = b') = p_{b'}$
32:                $\mathcal{R}_b = \mathcal{R}_k$
33: return $\mathcal{R}_{\text{best}}$
```

We note that Nikolić and Teodorović (2013) describe theirs as a "bee colony optimization" algorithm. As noted in Sörensen (2015), "bee colony optimization" is merely a relabelling of the components of one kind of evolutionary algorithm. It is mathematically identical to this older, well-established metaheuristic. To avoid confusion and the spread of unnecessary terminology, we here describe the algorithm as an evolutionary algorithm.

Our neural evolutionary algorithm (NEA) differs from EA only in that the type-1 mutator is replaced with a "neural mutator". This mutator selects a route $r \in \mathcal{R}_b$ at random, deletes it from $\mathcal{R}_b$, and then rolls out learned construction with π_θ starting from $\mathcal{R}_b \setminus r$. Because it starts from $\mathcal{R}_b \setminus r$, and $|\mathcal{R}_b \setminus r| = S$, this produces just one new route r', which then replaces r in $\mathcal{R}_b$. The algorithm is otherwise unchanged. We replace the type-1 mutator because its space of changes (replacing one route by a shortest path) is similar to that of the neural mutator (replacing one route by a new route composed of shortest paths), while the type-2 mutator's space of changes is quite different (lengthening or shortening a route by one node). In this way we leverage π_θ, which was trained as a construction policy, to aid in an improvement method.

5 MUMFORD EXPERIMENTS

We first evaluated our method on the Mandl (1980) and Mumford (2013b) city datasets, two popular benchmarks for evaluating NDP algorithms (Mumford, 2013a; John et al., 2014; Kılıç and Gök, 2014; Ahmed et al., 2019). The Mandl dataset is one small synthetic city, while the Mumford dataset consists of four synthetic cities, labelled Mumford0 through Mumford3, and gives values of S, MIN, and MAX to use when benchmarking on each city. The values n, S, MIN, and MAX for Mumford1, Mumford2, and Mumford3 are taken from three different real-world cities and their existing transit networks, giving the dataset a degree of realism. Details of these benchmarks, and the parameters we use when evaluating on them, are given in Table 1.

In all of our experiments, we set the transfer penalty p_T used to compute average trip time C_p to $p_T = 300$s (five minutes). This value is a reasonable estimate of a passenger's preference for avoiding transfers versus saving time; it is widely used by other methods when evaluating on these benchmarks (Mumford, 2013a), so using this same value of p_T allows us to directly compare our results with other results on these benchmarks.

5.1 Comparison with Baseline Evolutionary Algorithm

We run each algorithm under consideration on all five of these synthetic cities over eleven different values of α, ranging from 0.0 to 1.0 in increments of 0.1. This lets us observe how well the different methods perform under a range of possible preferences, from the extremes of the operator perspective ($\alpha = 0.0$, caring only about C_o) and passenger perspective ($\alpha = 1.0$, caring only about C_p) to a range of intermediate perspectives. The constraint weight is set as $\beta = 5.0$ in all experiments, the same value used in training the policies. We found that this was sufficient to prevent any of the constraints in section 3 from being violated by any transit network produced in our experiments.

Because these algorithms are stochastic, for each city we perform ten runs of each algorithm with ten different random seeds. For algorithms that make use of a learned policy, ten separate policies were trained with the same set of random seeds (but using the same training dataset), and each was used when running al-

Table 1: Statistics of the five benchmark problems used in our experiments.

City	# nodes n	# street edges $\lvert\mathcal{E}_s\rvert$	# routes S	MIN	MAX	Area (km^2)
Mandl	15	20	6	2	8	352.7
Mumford0	30	90	12	2	15	354.2
Mumford1	70	210	15	10	30	858.5
Mumford2	110	385	56	10	22	1394.3
Mumford3	127	425	60	12	25	1703.2

gorithms with the corresponding random seed. The values reported are statistics computed over the ten runs.

We first compared our neural evolutionary algorithm (NEA) with our baseline evolutionary algorithm (EA). We also compare both with the initial networks from LC-100, to see how much improvement each algorithm makes over the initial networks. In the EA and NEA runs, the same parameter settings of $B = 10, IT = 400, E = 10$ were used, following the values used in Nikolić and Teodorović (2013).

Table 2 displays the cost $C(\mathcal{C}, \mathcal{R})$ achieved by each algorithm on the Mandl and Mumford benchmarks for the operator perspective ($\alpha = 0.0$), the passenger perspective ($\alpha = 1.0$), and a balanced perspective ($\alpha = 0.5$). We see that on Mandl, the smallest of the five cities, EA and NEA perform virtually identically. But for the three largest cities, NEA performs considerably better than EA for both the operator and balanced perspectives. On Mumford3, the largest of the five cities, NEA solutions have 13% lower average cost for the operator perspective and 5% lower for the balanced perspective than EA. For the passenger perspective NEA's advantage over EA is smaller, but it still outperforms EA on all of the Mumford cities.

Figure 3 displays the average trip time C_p and total route time C_o achieved by each algorithm on the three largest Mumford cities, each of which are based on a real city's statistics, at eleven α values evenly spaced over the range $[0, 1]$ in increments of 0.1. There is a necessary trade-off between C_p and C_o, and as we would expect, C_o increases and C_p decreases for each algorithm's transit networks as α increases. We also observe that for $\alpha < 1.0$, NEA pushes C_o much lower than LC-100 and EA. NEA achieves an overall larger range of outcomes than LC-100 and EA - for any network $\mathcal{R}$ from LC-100, there is some value of α for which NEA produces a network $\mathcal{R}'$ that dominates $\mathcal{R}$. Figure 3 also shows results for an additional algorithm, LC-40k, which we discuss in subsection 5.2.

On all three of Mumford1 through Mumford3, we observe a common pattern: EA and NEA perform very similarly at $\alpha = 1.0$ (the leftmost point on each curve), but a significant performance gap forms as α decreases - consistent with what we see in Table 2. We also observe that LC-100 favours reducing C_p over C_o, with its points clustered higher and more leftwards than most of the points with corresponding α values on the other curves. Both EA and NEA achieve only very small decreases in C_p on LC-100's initial solutions at $\alpha = 1.0$, but much larger decreases in C_o at lower α.

5.2 Ablation Studies

To better understand the contribution of various components of our method, we performed three sets of ablation studies. These were conducted over the three realistic Mumford cities (1, 2, and 3) and over eleven α values evenly spaced over the range $[0, 1]$ in increments of 0.1.

5.2.1 Effect of number of samples

We note that over the course of the evolutionary algorithm, with our parameter settings $B = 10, IT = 400, E = 10$, a total of $B \times IT \times E = 40,000$ different transit networks are considered. By comparison, LC-100 only considers 100 networks. NEA's superiority to LC-100 may partly be due to its considering more networks. To test this, we ran LC-40k, in which we sample 40,000 networks from the learned-construction algorithm, and pick the lowest-cost network. Comparing LC-100 and LC-40k in Figure 3, we see that across all three cities and all values of α, LC-40k performs very similarly to LC-100, improving on it only slightly in comparison with the larger improvements given by EA or NEA. From this we conclude that the number of networks considered is not on its own an important factor in EA's and NEA's performance: much more important is the evolutionary algorithm that guides the search of possible networks.

5.2.2 Contribution of type-2 mutator

We next considered the impact of the type-2 mutator heuristic on NEA. This low-level heuristic is the same between EA and NEA and is not a learned function. To understand how important it is to NEA's performance,

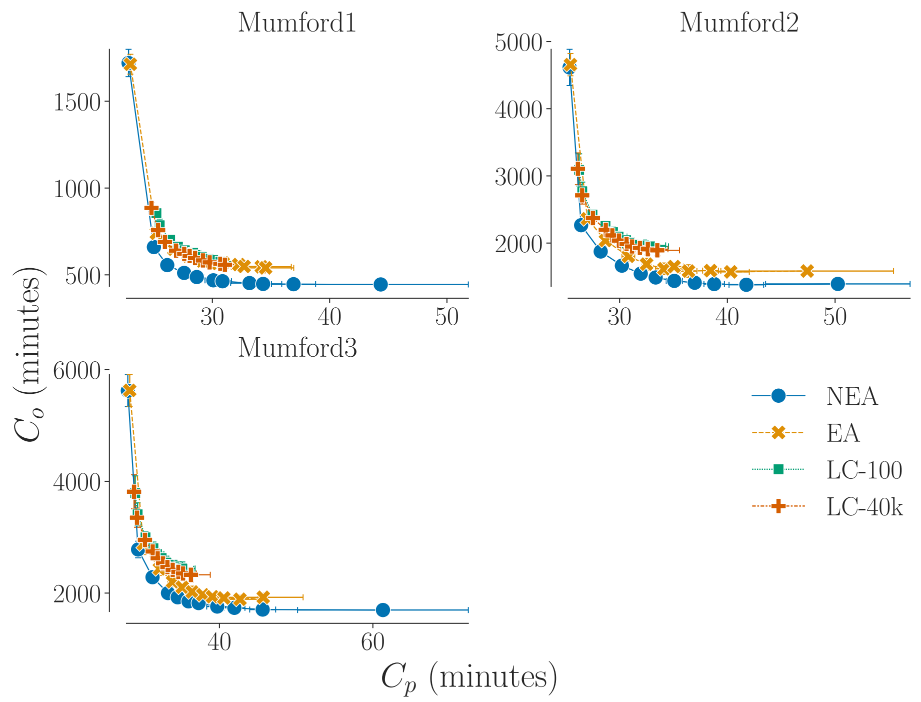


Fig. 3: Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis), across values of α over the range $[0, 1]$ in increments of 0.1, from $\alpha = 0.0$ at lower-right to 1.0 at upper-left, for transit networks from LC-100, LC-40k, EA, and NEA. Each point shows the mean C_p and C_o over 10 random seeds for one value of α, and bars around each point indicate one standard deviation on each axis. Lines linking pairs of points indicate that they represent consecutive α values. Lower values of C_o and C_p are better, so the down-and-leftward direction in each plot represents improvement.

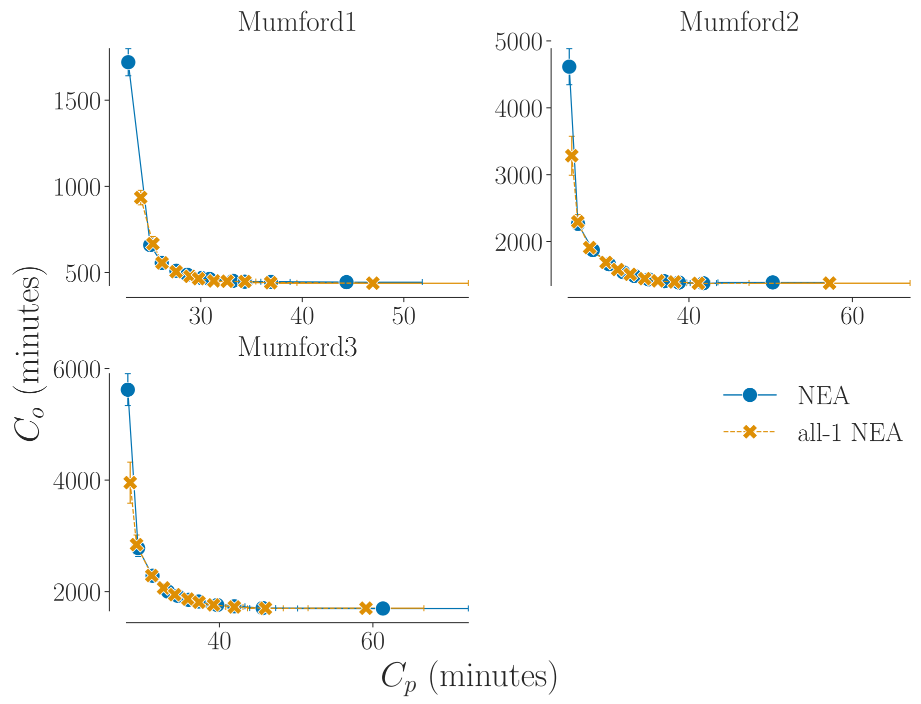


Fig. 4: Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis) achieved by all-1 NEA, plotted along with NEA (repeated from Figure 3) for comparison.

Table 2: Final cost $C(\alpha, \mathcal{C}, \mathcal{R})$ achieved baseline experiments for three different α values. Values are averaged over ten random seeds; the $\pm$ value is the standard deviation of $C(\alpha, \mathcal{C}, \mathcal{R})$ over the seeds.

α	Method	Mandl	Mumford0	Mumford1	Mumford2	Mumford3
0.0	LC-100	0.697 $\pm$ 0.011	0.847 $\pm$ 0.025	1.747 $\pm$ 0.034	1.315 $\pm$ 0.049	1.333 $\pm$ 0.064
	EA	**0.687 $\pm$ 0.016**	**0.776 $\pm$ 0.026**	1.637 $\pm$ 0.065	1.067 $\pm$ 0.032	1.052 $\pm$ 0.041
	NEA	**0.687 $\pm$ 0.016**	0.783 $\pm$ 0.027	**1.347 $\pm$ 0.033**	**0.938 $\pm$ 0.024**	**0.927 $\pm$ 0.014**
0.5	LC-100	0.558 $\pm$ 0.003	0.916 $\pm$ 0.006	1.272 $\pm$ 0.018	0.989 $\pm$ 0.021	0.984 $\pm$ 0.026
	EA	**0.549 $\pm$ 0.008**	0.841 $\pm$ 0.019	1.203 $\pm$ 0.026	0.866 $\pm$ 0.023	0.851 $\pm$ 0.025
	NEA	0.550 $\pm$ 0.006	**0.840 $\pm$ 0.021**	**1.052 $\pm$ 0.013**	**0.816 $\pm$ 0.008**	**0.801 $\pm$ 0.008**
1.0	LC-100	0.328 $\pm$ 0.001	0.721 $\pm$ 0.004	0.573 $\pm$ 0.004	0.495 $\pm$ 0.002	0.476 $\pm$ 0.001
	EA	**0.315 $\pm$ 0.001**	0.590 $\pm$ 0.005	0.523 $\pm$ 0.003	0.480 $\pm$ 0.001	0.464 $\pm$ 0.002
	NEA	**0.315 $\pm$ 0.001**	**0.587 $\pm$ 0.004**	**0.520 $\pm$ 0.004**	**0.479 $\pm$ 0.002**	**0.460 $\pm$ 0.003**

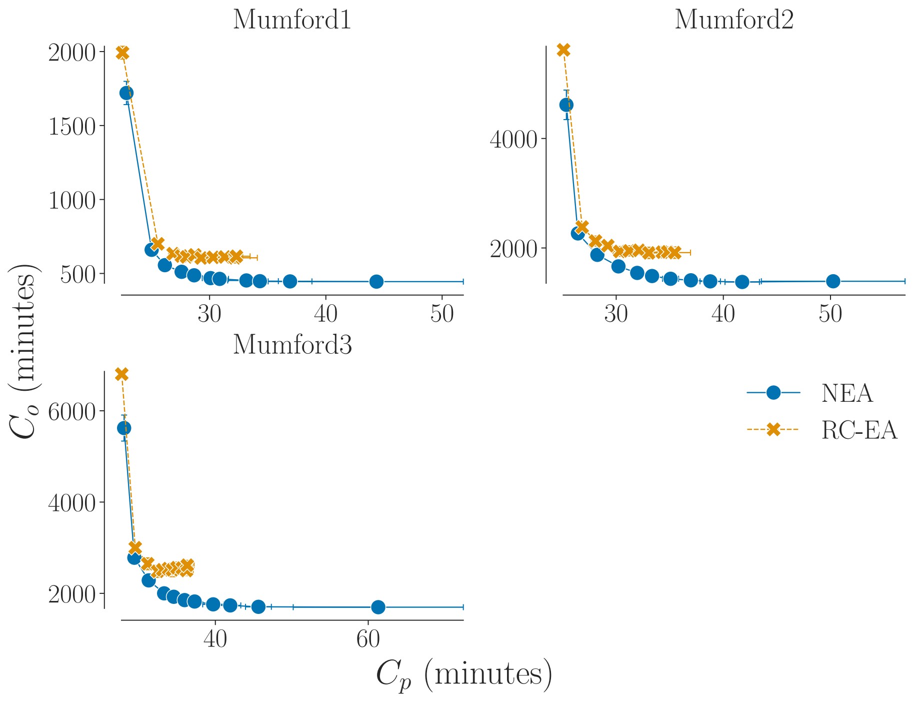


Fig. 5: Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis) achieved by RC-EA, plotted along with NEA (repeated from Figure 3) for comparison.

we ran “all-1 NEA”, a variant of NEA in which only the neural mutator is used, and not the type-2 mutator. The results are shown in Figure 4, along with the curve for NEA as in Figure 3, for comparison. It is clear that NEA and all-1 NEA perform very similarly in most scenarios. The main difference is at $\alpha = 1.0$, where we see that all-1 NEA underperforms NEA, achieving higher C_p. It appears that the minor adjustments to existing routes made by the type-2 mutator are more important here, where the neural net policy has been pushed to the extremes of its behaviour, but have little impact at $\alpha < 1.0$.

5.2.3 Importance of learned heuristics

We note that the type-1 mutator used in EA and the neural mutator used in NEA differ in the space of changes each is capable of making. The type-1 mutator can only add shortest paths as routes, while the neural mutator may compose multiple shortest paths into its new routes. It may be that this structural difference, as opposed to the quality of the heuristic learned by the policy π_θ, is part of NEA's advantage over EA.

To test this conjecture, we ran a variant of NEA in which the learned policy π_θ is replaced by a policy

π_{random} that chooses actions uniformly at random:

$$\pi_{\text{random}}(a|s_t) = \frac{1}{|\mathcal{A}_t|} \; \forall \; a \in \mathcal{A}_t$$

We call this variant the randomly-combining evolutionary algorithm (RC-EA). Since we wanted to gauge the performance of this variant in isolation from the learned policy, we used π_{random} in the LC-100 algorithm to generate $\mathcal{R}_0$ for RC-EA.

The results are shown in Figure 5, along with the same NEA curves as in Figure 3. For values of $\alpha < 1.0$, RC-EA performs less well than both EA and NEA. But at the extreme of $\alpha = 1.0$ it performs better than NEA, decreasing average trip time C_p by about twenty seconds versus NEA - an improvement of about 1% - on each city.

RC-EA's poor performance at low α makes sense: its ability to replace routes with composites of shortest paths biases it towards making longer routes, and unlike π_θ, π_{random} cannot learn to prefer halting early in such cases. This is an advantage over NEA at $\alpha = 1.0$ but a disadvantage at $\alpha < 1.0$. It is interesting, though, that RC-EA outperforms NEA at $\alpha = 1.0$, as these two share the same space of possible actions. We would expect that the neural net policy used in NEA should be able to learn to reach that performance of RC-EA simply by learning to choose actions uniformly at random, yet here it fails to do so.

Nonetheless, NEA performs better overall than RC-EA, and so we leave answering this question to future work. For now we merely note that composing shortest paths randomly is a good heuristic for route construction at $\alpha = 1$, but not below that; so the advantage of NEA comes primarily from what the policy π_θ learns during training.

5.3 Comparisons with prior work

While conducting the experiments of subsection 5.1, we observed that after 400 iterations the cost of NEA's best-so-far network $\mathcal{R}_{\text{best}}$ was still decreasing from one iteration to the next. In order to make a fairer comparison with other methods from the literature on the Mandl and Mumford benchmarks, we performed a further set of experiments where we ran NEA with $IT = 4,000$ instead of 400, for the operator perspective ($\alpha = 0.0$) and the passenger perspective ($\alpha = 1.0$). In addition, because we observed better performance from RC-EA for $\alpha = 1.0$, we also ran RC-EA with $IT = 4,000$ for $\alpha = 1.0$, and we include these results in the comparison.

Table 3 and Table 4 present the results of these experiments on the Mandl and Mumford benchmarks, alongside results reported on these benchmarks in comparable recent work. As elsewhere, the results we report for NEA and RC-EA are averaged over ten runs with different random seeds; the same set of ten trained policies used in the experiments of subsection 5.1 and subsection 5.2 are used here in NEA. In addition to the average trip time C_p and total route time C_o, these tables present metrics of how many transfers between routes were required by the networks. These are labeled d_0, d_1, d_2, and d_{un}. d_i with $i \in \{0, 1, 2\}$ is the percentage of all passenger trips that required i transfers between routes; while d_{un} is the percentage of trips that require more than two transfers: $d_{un} = 100 - (d_0 + d_1 + d_2)$.

Before discussing these results, a brief word should be said about computation time. On a desktop computer with a 2.4 GHz Intel i9-12900F processor and an NVIDIA RTX 3090 graphics processing unit (used to accelerate our neural net computations), NEA takes about 10 hours for each 4,000-iteration run on Mumford3, the largest environment, while the RC-EA runs take about 6 hours. By comparison, John et al. (2014) and Hüsselmann et al. (2023) both use a variant of NSGA-II, a genetic algorithm, with a population of 200 networks, which in both cases they run for more than two days on Mumford3. Hüsselmann et al. (2023)'s DB-MOSA variant, meanwhile, takes 7 hours and 52 minutes to run on Mumford3. Kılıç and Gök (2014) report that their procedure takes eight hours just to construct the initial network for Mumford3, and don't report the running time for the subsequent optimization.

These reported running times are not directly comparable as the experiments were not run on identical hardware, but they are broadly indicative of the differences in speed of these methods. The running times are mainly a function of the parameters of the metaheuristic algorithm used, rather than the low-level heuristics used in the algorithm. The large populations used in some of these methods make them very time-consuming because each network in the population must be modified and evaluated at each step. But their search through solution space is more exhaustive than single-solution methods such as simulated annealing, or an evolutionary algorithm with a small population ($B = 10$) as we use in our own experiments. We are mainly interested here in the quality of our low-level heuristics, rather than that of the metaheuristic algorithm.

5.3.1 Results

Table 3 shows the passenger perspective results alongside results from other work. On Mandl, Mumford0, and Mumford1, the hyper-heuristic method of Ahmed

City	Method	$C_p \downarrow$	C_o	$d_0 \uparrow$	d_1	d_2	$d_{un} \downarrow$
Mandl	Mumford (2013a)	10.27	221	95.38	4.56	0.06	**0**
	John et al. (2014)	10.25	212	-	-	-	-
	Kılıç and Gök (2014)	10.29	216	95.5	4.5	0	**0**
	Ahmed et al. (2019)	**10.18**	212	97.17	2.82	0.00	**0**
	Hüsselmann et al. (2023) DBMOSA	10.27	179	95.94	3.93	0.13	**0**
	Hüsselmann et al. (2023) NSGA-II	10.19	197	**97.36**	2.64	0	**0**
	NEA	10.37	181	93.89	5.93	0.18	**0**
	RC-EA	10.27	193	95.83	4.16	0.01	**0**
Mumford0	Mumford (2013a)	16.05	759	63.2	35.82	0.98	**0**
	John et al. (2014)	15.4	745	-	-	-	-
	Kılıç and Gök (2014)	14.99	707	69.73	30.03	0.24	**0**
	Ahmed et al. (2019)	**14.09**	722	**88.74**	11.25	0	**0**
	Hüsselmann et al. (2023) DBMOSA	15.48	431	65.5	34.5	0	**0**
	Hüsselmann et al. (2023) NSGA-II	14.34	635	86.94	13.06	0	**0**
	NEA	15.26	639	68.35	31.24	0.41	**0**
	RC-EA	14.62	735	77.87	22.13	0.00	**0**
Mumford1	Mumford (2013a)	24.79	2038	36.6	52.42	10.71	0.26
	John et al. (2014)	23.91	1861	-	-	-	-
	Kılıç and Gök (2014)	23.25	1956	45.1	49.08	5.76	0.06
	Ahmed et al. (2019)	**21.69**	1956	**65.75**	34.18	0.07	**0**
	Hüsselmann et al. (2023) DBMOSA	22.31	1359	57.14	42.63	0.23	**0**
	Hüsselmann et al. (2023) NSGA-II	21.94	1851	62.11	37.84	0.05	**0**
	NEA	22.85	1723	49.28	48.94	1.78	**0**
	RC-EA	22.29	2032	53.47	45.96	0.57	**0**
Mumford2	Mumford (2013a)	28.65	5632	30.92	51.29	16.36	1.44
	John et al. (2014)	27.02	5461	-	-	-	-
	Kılıç and Gök (2014)	26.82	5027	33.88	57.18	8.77	0.17
	Ahmed et al. (2019)	25.19	5257	**56.68**	43.26	0.05	**0**
	Hüsselmann et al. (2023) DBMOSA	25.65	3583	48.07	51.29	0.64	**0**
	Hüsselmann et al. (2023) NSGA-II	25.31	4171	52.56	47.33	0.11	**0**
	NEA	25.25	4937	52.66	47.05	0.29	**0**
	RC-EA	**24.92**	5655	54.12	45.85	0.04	**0**
Mumford3	Mumford (2013a)	31.44	6665	27.46	50.97	18.79	2.81
	John et al. (2014)	29.5	6320	-	-	-	-
	Kılıç and Gök (2014)	30.41	5834	27.56	53.25	17.51	1.68
	Ahmed et al. (2019)	28.05	6119	50.41	48.81	0.77	**0**
	Hüsselmann et al. (2023) DBMOSA	28.22	4060	45.07	54.37	0.56	**0**
	Hüsselmann et al. (2023) NSGA-II	28.03	5018	48.71	51.1	0.19	**0**
	NEA	27.96	6127	49.36	50.04	0.60	**0**
	RC-EA	**27.60**	6896	**50.90**	48.99	0.11	**0**

Table 3: Passenger-perspective results. C_p is the average passenger trip time. C_o is the total route time. d_i is the percentage of trips satisfied with number of transfers i, while d_{un} is the percentage of trips satisfied with 3 or more transfers. Arrows next to each quantity indicate which of increase or decrease is desirable. Bolded values in C_p, d_0, and d_{un} columns are the best on that environment.

et al. (2019) performs best. But on Mumford3 - the most challenging benchmark city - our NEA algorithm outperforms all other methods in the literature, improving on the previous best of Hüsselmann et al. (2023)'s NSGA-II by 0.3%. Meanwhile, RC-EA performs best of all on both Mumford2 and Mumford3, improving on Mumford3's previous best result by 1.5%, and setting a new state-of-the-art on these two challenging benchmarks.

Table 4 shows the operator-perspective results alongside results from other work. Kılıç and Gök (2014) do not report results for the operator perspective and as such we do not include their work in Table 4. Similarly to the passenger-perspective results, NEA underperforms on the smallest cities (Mandl and Mumford0), but outperforms all methods other than Ahmed et al. (2019) on Mumford1 and Mumford2, including the more recent work of Hüsselmann et al. (2023). And on Mumford3, NEA again outperforms all other methods, improving on the previous state of the art by 4.8%.

We see the same pattern here as in subsection 5.1, where NEA's performance relative to other methods improves as the city graph grows larger. NEA's performance gap with the previous state of the art narrows consistently over the Mumford cities as they grow in size, until surpassing it on Mumford3 in both the passenger and operator perspectives. This is the case despite NEA and RC-EA's relatively under-powered metaheuristic. This shows that for large city graphs, our learned policy functions very well as a low-level heuristic across all values of α. And in the case of $\alpha = 1.0$, our un-learned heuristic of randomly concatenating shortest paths performs even better. The pattern in these results also suggest that our methods' advantages will continue to grow with the size of the city graph.

The metrics d_0 to d_{un} reveal that at $\alpha = 0.0$, NEA favours higher numbers of transfers relative to most of the other methods, particularly Hüsselmann et al. (2023). This matches our intuitions: shorter routes will deliver fewer passengers directly to their destinations and will require more transfers, so it is logical that the learned policies would tend to increase the number of transfers when rewarded for minimizing operator cost. Meanwhile, at $\alpha = 1.0$, the numbers of transfers achieved by NEA and RC-EA are comparable to other recent methods; both methods avoid having passengers make any trips with more than two transfers.

6 LAVAL EXPERIMENTS

To assess our method in a realistic problem instance, we applied it to data from the city of Laval in Quebec, Canada. Laval is a suburb of the major Canadian city of Montreal. As of the 2021 census, it had a total population of 429,555 (Statistics Canada, 2023). Public transit in Laval is provided primarily by the bus network of the Société de Transport de Laval; additionally, one line of the Montreal underground metro system has three stops in Laval.

6.1 Representing Laval

To apply our learned policy π_θ to Laval, we first had to assemble a realistic city graph $\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D)$ representing Laval from the data sources that were available to us. Our model of the city of Laval is based on several sources: geographic data on census dissemination areas (CDAs) for 2021 from Statistics Canada (Statistics Canada, 2021), a GIS representation of the road network of Laval provided to us by the Société de Transport de Laval, an OD dataset (Agence métropolitaine de transport, 2013) provided by Quebec's Agence Métropolitaine de Transport, and publicly-available GTFS data from 2013 (STL) that describes Laval's existing transit system.

6.1.1 Street graph

The real city of Laval contains 2,618 unique transit stop facilities. Taking each of these to be a node would yield too large a graph for our neural policy to be able to process on the commercial GPU hardware available to us - the GPU memory requirements would simply be too great. It is also likely that such a representation would be more fine-grained than necessary: the demand for travel within the service areas of nearby transit stops is likely to be very similar. For these reasons, we instead made use of a coarsened graph, where the nodes correspond to CDAs. CDAs are designed so as to enclose populations that are roughly homogeneous (among other factors), making them a sensible granularity at which to analyze travel demands. Laval contains 632 distinct CDAs, and a graph size of $n = 632$ makes it feasible for us to apply our neural policy with the hardware we have available.

The street graph $(\mathcal{N}, \mathcal{E}_s)$ was derived from the 2021 census data by taking the centroid of each CDA within Laval to be a node in $\mathcal{N}$, and adding a street edge (i, j, t_{ij}) for node pair (i, j) if their corresponding CDAs shared a border. To compute drive times t_{ij}, we found all points in the road network within the CDAs of i and j, computed the shortest-path driving time over the road network from each of i's road-network points to each of j's, and set t'_{ij} as the median of these driving times. We did this so that t'_{ij} would reflect the real drive times given the existing road network. By this approach, $t'_{ij} \neq t'_{ji}$ in general, but as our method treats cities as undirected graphs, it expects that $t_{ij} = t_{ji} \; \forall \, i, j \in \mathcal{N}$. To enforce this, as a final step we set:

$$t_{ij} = t_{ji} = \max(t'_{ij}, t'_{ji}) \; \forall \, i, j \in \mathcal{N} \tag{13}$$

6.1.2 Existing transit

The existing transit network in Laval has 43 bus routes that operate during morning rush hour from 7 to 9 AM. To compare networks from our algorithm to the this network, we had to translate it to a network that runs over $(\mathcal{N}, \mathcal{E}_s)$ as defined in subsection 6.1.1. To do this, we mapped each stop on an existing bus route to a node in $\mathcal{N}$ based on which CDA contains the stop. We will refer to this translated transit system as the STL network, after the city's transit agency, the Société de Transport de Laval.

The numbers of stops on the routes in the STL network range from 2 to 52. To ensure a fair comparison,

City	Method	$C_o \downarrow$	C_p	$d_0 \uparrow$	d_1	d_2	$d_{un} \downarrow$
Mandl	Mumford (2013a)	**63**	15.13	70.91	25.5	2.95	0.64
	John et al. (2014)	**63**	13.48	-	-	-	-
	Ahmed et al. (2019)	**63**	14.28	62.23	27.16	9.57	1.03
	Hüsselmann et al. (2023) DBMOSA	**63**	13.55	70.99	24.44	4.00	**0.58**
	Hüsselmann et al. (2023) NSGA-II	**63**	13.49	**71.18**	25.21	2.97	0.64
	NEA	68	13.89	57.87	33.77	8.20	0.15
Mumford0	Mumford (2013a)	111	32.4	18.42	23.4	20.78	37.40
	John et al. (2014)	95	32.78	-	-	-	-
	Ahmed et al. (2019)	**94**	26.32	14.61	31.59	36.41	17.37
	Hüsselmann et al. (2023) DBMOSA	98	27.61	22.39	31.27	18.82	27.51
	Hüsselmann et al. (2023) NSGA-II	**94**	27.17	**24.71**	38.31	26.77	**10.22**
	NEA	122	30.20	15.34	30.77	27.82	26.08
Mumford1	Mumford (2013a)	568	34.69	16.53	29.06	29.93	24.66
	John et al. (2014)	462	39.98	-	-	-	-
	Ahmed et al. (2019)	**408**	39.45	18.02	29.88	31.9	20.19
	Hüsselmann et al. (2023) DBMOSA	511	26.48	**25.17**	59.33	14.54	**0.96**
	Hüsselmann et al. (2023) NSGA-II	465	31.26	19.70	42.09	33.87	4.33
	NEA	434	47.07	17.24	26.85	26.45	29.46
Mumford2	Mumford (2013a)	2244	36.54	13.76	27.69	29.53	29.02
	John et al. (2014)	1875	32.33	-	-	-	-
	Ahmed et al. (2019)	**1330**	46.86	13.63	23.58	23.94	38.82
	Hüsselmann et al. (2023) DBMOSA	1979	29.91	**22.77**	58.65	18.01	**0.57**
	Hüsselmann et al. (2023) NSGA-II	1545	37.52	13.48	36.79	34.33	15.39
	NEA	1356	54.82	10.99	22.41	27.00	39.60
Mumford3	Mumford (2013a)	2830	36.92	16.71	33.69	33.69	20.42
	John et al. (2014)	2301	36.12	-	-	-	-
	Ahmed et al. (2019)	1746	46.05	16.28	24.87	26.34	32.44
	Hüsselmann et al. (2023) DBMOSA	2682	32.33	**23.55**	58.05	17.18	**1.23**
	Hüsselmann et al. (2023) NSGA-II	2043	35.97	15.02	48.66	31.83	4.49
	NEA	**1663**	61.25	11.48	19.86	25.07	43.59

Table 4: Operator perspective results. C_p is the average passenger trip time. C_o is the total route time. d_i is the percentage of trips satisfied with number of transfers i, while d_{un} is the percentage of trips satsified with 3 or more transfers. Arrows next to each quantity indicate which of increase or decrease is desirable. Bolded values in C_o, d_0 and d_{un} columns are the best on that environment.

we set $MIN = 2$ and $MAX = 52$ when running our algorithm. Unlike the routes in our algorithm's networks, many of STL's routes are not symmetric (that is, they follow a different path in each direction between terminals), and several are unidirectional (they go only one way between terminals). We maintained the constraints of symmetry and bidirectionality on our own algorithm, but did not enforce them upon the STL network. The only modification we made is that when reporting total route time C_o in this section, we calculated it for bidirectional routes by summing the travel times of both directions, instead of just one direction as in the preceding sections.

In addition, there is an underground metro line, the Montreal Orange Line, which has three stops in Laval. As with the existing bus lines, we mapped these three stops to their containing CDAs and treat them as forming an additional route. We added this metro route to both the STL network and the networks $\mathcal{R}$ produced by our algorithm when evaluating them, to reflect that unlike the bus routes, it is not feasible to change the metro line and so it represents a constant of the city.

6.1.3 Demand matrix

Finally we assembled the demand matrix D from the OD dataset. The entries in the OD dataset correspond to trips reported by residents of Laval who were surveyed about their recent travel behaviour. Each entry has (lat,lon) coordinates l_o and l_d for the trip's approximate origin and destination, as well as an "expansion factor" e giving the estimated number of actual trips corresponding to that surveyed trip. It also indicates the mode of travel, such as car, bicycle, or public transit, that was used to make the trip.

Many entries in the OD dataset refer to trips that begin or end in Montreal. For our purposes, we "redirected" all trips made by public transit that enter or leave Laval to one of several "crossover points" that we defined. These crossover points are the locations of the three Orange Line metro stations in Laval, and the locations of the last or first stop in Laval of each existing bus route that goes between Laval and Montreal.

For each trip between Laval and Montreal, we identified the crossover point that has the shortest distance to either of l_o or l_d, and overwrote the Montreal endpoint of the trip by this crossover point. This process was automated with a simple computer script. The idea is that if the Laval end of the trip is close to a transit stop that provides access to Montreal, the rider will choose to cross over to the Montreal transit system as soon as possible, as most locations in Montreal will be easier to access once in the Montreal transit system. Trips that go between Laval and Montreal by means other than public transit were not included in D, as we judge that most such trips cannot be induced to switch modes to transit.

Having done this remapping, we initialized D with all entries set to 0. Then for each entry in the OD dataset, we found the CDAs that contain l_o and l_d, associated them with the matching nodes i and j in $\mathcal{N}$, and updated $D_{ij} \leftarrow D_{ij} + e$. D then had the estimated demand between every pair of CDAs. To enforce symmetric demand, we then assigned $D_{ij} \leftarrow \max(D_{ij}, D_{ji}) \; \forall \; i, j \in \mathcal{N}$. The resulting demand matrix D contains 548,159 trips, of which 63,104 are trips between Laval and Montreal that we redirected.

We then applied a final filtering step. The STL network does not provide a path between all node-pairs i, j for which $D_{ij} > 0$, and so it violates constraint 1 of section 3. This is because D at this point includes all trips that were made within Laval by any mode of transport, including to and from areas that are not served by the STL network. These areas may be unserved because they are populated by car owners unlikely to use transit if it were available. To ensure a fair comparison between the STL network and our algorithm's networks, we set to 0 all entries of D for which the STL network does not provide a path. We expected this to cause our system to output transit networks that are "closer" to the existing transit network, in that they satisfy the same travel demand as before. This should reduce the scope of the changes required to go from the existing network to a new one proposed by our system.

Table 5 contains the statistics of, and parameters used for, the Laval scenario.

6.2 Enforcing constraint satisfaction

In the experiments on the Mandl and Mumford benchmark cities described in section 5, the EA and NEA algorithms never produced networks violated any of the constraints on networks outlined in section 3. However, Laval's city graph, with 632 nodes, is considerably larger than even the largest Mumford city, which has only 127 nodes. In our initial experiments on Laval, we found that both EA and NEA consistently produced networks that violated constraint 1, providing no transit path for some node pairs (i, j) for which $D_{ij} > 0$.

To remedy this, we modified the MDP described in subsection 3.2 to enforce reduction of unsatisfied demand. Let $c_1(\mathcal{C}, \mathcal{R})$ be the number of node-pairs i, j with $D_{ij} > 0$ for which network $\mathcal{R}$ provides no transit path, and let $\mathcal{R}'_t = \mathcal{R}_t \cup \{r_t\}$, the network formed from the finished routes and the in-progress route r_t. We applied the following changes to the action space $\mathcal{A}_t$ whenever $c_1(\mathcal{R}_t) > 0$:

– When the timestep t is even and $\mathcal{A}_t$ would otherwise be $\{\text{continue}, \text{halt}\}$, the halt action is removed from $\mathcal{A}_t$. This means that if $c_1(\mathcal{C}, \mathcal{R}'_t) > 0$, the current route must be extended if it is possible to do so without violating another constraint.
– When t is odd, if $\mathcal{A}_t$ contains any paths that would reduce $c_1(\mathcal{C}, \mathcal{R}'_t)$ if added to r_t, then all paths that would *not* reduce $c_1(\mathcal{C}, \mathcal{R}'_t)$ are removed from $\mathcal{A}_t$. This means that if it is *possible* to connect some unconnected trips by extending r_t, then *not* doing so is forbidden.

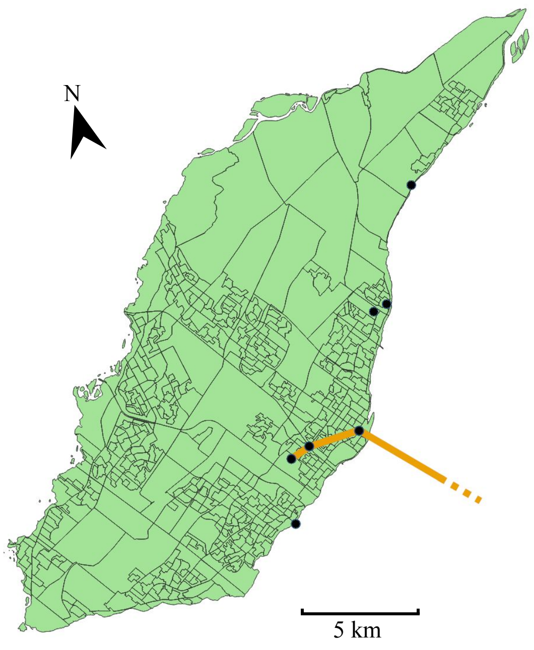
With these changes to the MDP, we found that both LC-100 and NEA produced networks that connected all desired passenger trips, with no constraint violations - even when the policy π_θ was trained without these changes. The results reported in this section are obtained using this variant of the MDP.

6.3 Experiments

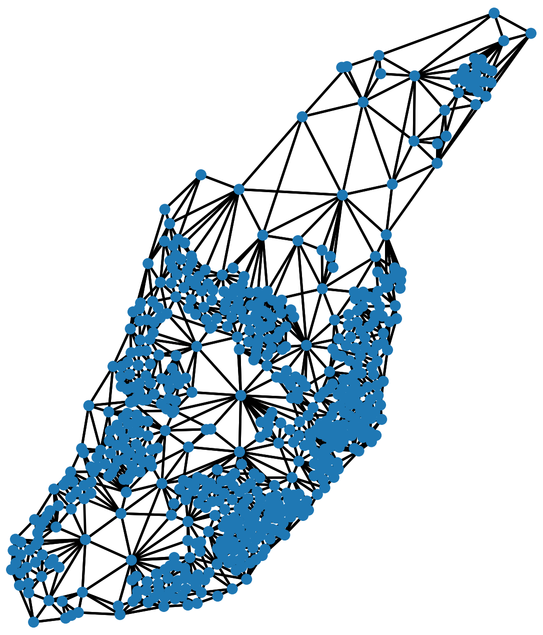
We performed experiments evaluating both the NEA and RC-EA algorithms on the Laval city graph. Three sets of NEA experiments were performed with different α values representing different perspectives: the operator perspective ($\alpha = 0.0$), the passenger perspective ($\alpha = 1.0$), and a balanced perspective ($\alpha = 0.5$). RC-EA experiments are only run at $\alpha = 1.0$ since we observed in subsection 5.2.3 that RC-EA did not perform well at other α. As in section 5, we perform ten runs of each experiment with ten different random seeds and ten different learned policies π_θ, and report statistics over these ten runs; the same per-seed parameters θ are used as in the experiments of section 5. We also run the STL network on the Laval city graph $\mathcal{C}$ in the same way that we run the networks from our other algorithms to measure their performance, but since the

Table 5: Statistics and parameters of the Laval scenario

# nodes n	# street edges $\lvert\mathcal{E}_s\rvert$	# demand trips	# routes S	MIN	MAX	Area (km^2)
632	4,544	548,159	43	2	52	520.1



(a) Census dissemination areas



(b) Street graph

Fig. 6: Figure 6a shows a map of the census dissemination areas of the city of Laval, with "crossover points" used to remap inter-city demands shown as black circles, and the Montreal Metro Orange Line shown in orange. Figure 6b shows the street graph constructed from these dissemination areas.

STL network is a single, pre-determined network that does not depend on α, we run it only once.

The parameters of NEA were the same as in the Mumford experiments in subsection 5.1: $IT = 400$, $B = 10$, $E = 10$, and transfer penalty $p_T = 300$s. While the memory requirements of running our method on this very large graph were greater than for the Mumford benchmark datasets, we were still able to run the experiments on the desktop machine used for our Mumford experiments, without any changes to our model or algorithm code.

6.3.1 Results

The results of the Laval experiments are shown in Table 6. The results for the initial networks produced by LC-100 are presented as well, to show how much improvement the NEA makes over its initial networks. Figure 7 highlights the trade-offs each method achieves between average trip time C_p and total route time C_o, and how they compare to the STL network.

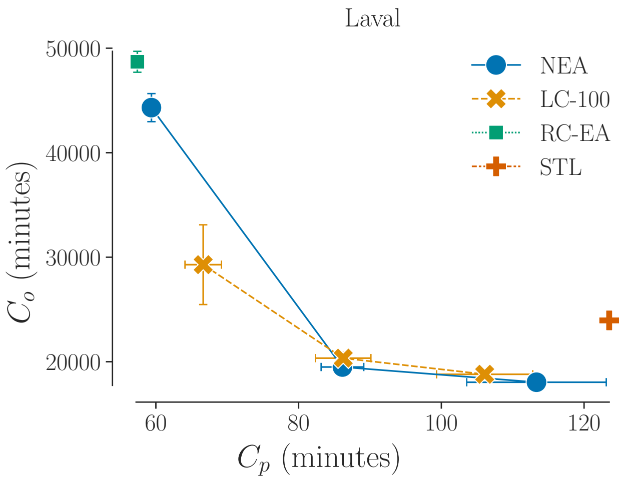


Fig. 7: Average trip time C_p versus total route time C_o achieved by LC-100 and NEA at $\alpha = 0.0$ (bottom-right), 0.5 (middle), and 1.0 (top-left), as well as for RC-EA at $\alpha = 1.0$ and the STL network itself (independent of α). Except for STL, each point is a mean value over 10 random seeds, and bars show one standard deviation.

α	Method	$C_p \downarrow$	$C_o \downarrow$	$d_0 \uparrow$	d_1	d_2	$d_{un} \downarrow$
N/A	STL	123.52	23954	15.75	25.86	22.87	35.51
0.0	LC-100	**106.08 ± 6.74**	18797 ± 190	15.45 ± 0.54	22.98 ± 1.34	23.71 ± 1.08	37.86 ± 1.87
	NEA	113.33 ± 9.78	**18037 ± 302**	15.44 ± 0.47	22.50 ± 1.72	23.23 ± 1.27	38.83 ± 2.67
0.5	LC-100	86.25 ± 3.88	20341 ± 549	15.04 ± 0.46	26.46 ± 1.10	26.76 ± 1.18	**31.74 ± 2.22**
	NEA	**86.14 ± 2.99**	**19494 ± 389**	14.89 ± 0.62	26.18 ± 1.53	26.59 ± 1.69	32.34 ± 2.90
1.0	LC-100	66.64 ± 2.56	**29286 ± 3815**	17.16 ± 1.27	34.50 ± 3.49	29.03 ± 1.00	19.30 ± 4.81
	NEA	59.38 ± 0.52	44314 ± 1344	22.08 ± 0.71	38.79 ± 2.03	27.57 ± 0.75	11.56 ± 2.22
	RC-EA	**57.40 ± 0.17**	48700 ± 1001	**23.73 ± 0.57**	44.31 ± 0.71	25.20 ± 0.62	**6.76 ± 0.58**

Table 6: Performance of networks from LC-100 and NEA at three α values, RC-EA at $\alpha = 1.0$, and the STL network. Bolded values are best for the corresponding α, except where STL performs best at that α, in which case no values are bolded. C_p is the average passenger trip time. C_o is the total route time. d_i is the percentage of trips satisfied with i transfers, while d_{un} is the percentage of trips satisfied with 3 or more transfers. All values are averaged over ten random seeds, with one standard deviation following "±" where relevant.

We see that for each value of α, NEA's network outperforms the STL network: it achieves 52% lower passenger cost C_p at $\alpha = 1.0$, 25% lower operator cost C_o at $\alpha = 0.0$, and at $\alpha = 0.5$, 30% lower C_p and 19% lower C_o. At both $\alpha = 0.0$ and $\alpha = 0.5$, NEA dominates the existing transit system, achieving both lower C_p and lower C_o.

Comparing NEA's results to LC-100, we see that at $\alpha = 0.0$, NEA decreases C_o and increases C_p relative to LC-100; and for $\alpha = 1.0$, the reverse is true. In both cases, NEA improves over the initial network on the objective being optimized, at the cost of the other, as we would expect. In the balanced case ($\alpha = 0.5$), we see that NEA decreases C_o by 4% and C_p by 0.1% relative to LC-100. This matches what we observed in Figure 3, as discussed in subsection 5.1, that NEA's improvements over LC-100 tend to be greater in C_o than in C_p. RC-EA, meanwhile, improves on NEA at $\alpha = 1.0$ in terms of C_p, as it did in our other experiments. Here, it decreases C_p by two minutes on average, or about 3.3%.

Looking at the transfer percentages d_0, d_1, d_2, and d_{un}, we see that as α increases, the overall number of transfers shrinks, with d_0, d_1 and d_2 growing and d_{un} shrinking. This is as expected, since fewer transfers implies reduced passenger travel times. By the same token, at $\alpha = 1.0$ RC-EA performs best on these metrics, as it does on C_p, and NEA here outperforms the STL network by these metrics as well. At $\alpha = 0.5$, surprisingly, NEA's d_0 is lower than at the other α values, and lower than the STL network's d_0 by 0.86% of trips. But NEA's d_{un} is lower than STL's by 3.2% of trips, while d_1 and d_2 are both higher, meaning that NEA's networks let more passengers reach their destination in 2 or fewer transfers than the STL network. It is only at $\alpha = 0.0$ that STL's network requires fewer transfers overall than NEA's networks, and this is reasonable given that at $\alpha = 0.0$ the goal is strictly to minimize route length regardless of transfers.

The likely aim of Laval's transit agency is to reduce costs while improving service quality, or at least not harming it too much. The "balanced" case, where $\alpha = 0.5$, is most relevant to that aim. In this balanced case, we find that NEA proposes networks that reduce total route time by 19% versus the existing STL transit network. Some simple calculations can help understand the savings this represents. The approximate cost of operating a Laval city bus is 200 Canadian dollars (CAD) per hour. Suppose that each route has the same headway (time between bus arrivals) H. Then, the number of buses N_r required by route r is the ceiling of the route's driving time τ_r divided by the headway, $\lceil \frac{\tau_r}{H} \rceil$. The total number of buses $N_\mathcal{R}$ required for a transit network $\mathcal{R}$ is then:

$$N_\mathcal{R} = \sum_{r \in \mathcal{R}} N_r \approx \frac{1}{H} \sum_{r \in \mathcal{R}} \tau_r = \frac{C_o}{H}$$

The total cost of the system is 200 CAD$\times N_\mathcal{R}$. Assuming a headway of 15 minutes on all routes, that gives a per-hour operating cost of 319,387 CAD for the STL network, while the average operating cost of networks from NEA with $\alpha = 0.5$ is 259,920 CAD. So, under these simplifying assumptions, NEA could save the Laval transit agency on the order of 59,500 CAD per hour, a 19% reduction in operating costs. This is only a rough estimate: in practice, headways differ between routes, and there may be additional costs involved in altering the routes from the current system, such as from moving or building bus stops. But it shows that NEA, and neural heuristics more broadly, may offer practical savings to transit agencies.

In addition, NEA's balanced-case networks reduce the average passenger trip time by 33% versus the STL network, and reduce the number of transfers that passengers have to make, especially reducing the percentage of trips that require three or more transfers from 41.83% to 37.08%. Trips requiring three or more transfers are widely regarded as being so unattractive that riders will not make them (but note that these trips are still included in the calculation of C_p). So NEA's networks may even increase overall transit ridership. This not only makes the system more useful to the city's residents, but also increases the agency's revenue from fares.

In these experiments, the nodes of the graph represent census dissemination areas instead of existing bus stops. In order to be used in reality, the proposed routes would need to be fitted to the existing locations of bus stops in the CDAs, likely making multiple stops within each CDAs. To minimize the cost of the network redesign, it would be necessary to keep stops at most or all stop locations that have shelters, as constructing or moving these may cost tens of thousands of dollars, while stops with only a signpost at their locations can be moved at the cost of only a few thousand dollars. An algorithm for translating our census-level routes to real-stop-level routes, in a way that minimizes cost, is an important next step if the methods developed here are to be applied in real cities. We leave this for future work.

7 CONCLUSIONS

The choice of low-level heuristics has a major impact on the effectiveness of a metaheuristic algorithm, and our results show that using deep reinforcement learning to learn heuristics can offer substantial benefits when used alongside human-engineered heuristics. They also show that learned heuristics may be useful in complex real-world transit planning scenarios. Learned heuristics can improve on an existing transit system in multiple dimensions, in a way sensitive to the planner's preference over these dimensions, and so may enable transit agencies to offer better service at reduced cost.

Our heuristic-learning method has several limitations. One is that the construction MDP on which our neural net policies are trained differs from the evolutionary algorithm - an improvement process - in which it is deployed. Another is that it learns just one type of heuristic, that for constructing a route from shortest paths, though we note that the neural net may be learning multiple distinct heuristics for how to do this under different circumstances. Natural next steps would be to learn a more diverse set of heuristics, perhaps including route-lengthening and -shortening operators, and to train our neural net policies directly in the context of an improvement process, which may yield learned heuristics that are better-suited to that process.

A remaining question is why the random path-combining heuristic used in the RC-EA experiments outperforms the learned heuristic in the extreme of the passenger perspective. As noted in subsection 5.2.3, in principle the neural net should be able to learn whatever policy the random path-combiner is enacting: in the limit, it could learn to give the same probability to all actions when $\alpha = 1.0$. We wish to explore in more depth why this does not occur, and whether changes to the policy architecture or learning algorithm might allow the learned heuristic's performance to match or exceed that of the random policy.

The evolutionary algorithm in which we use our learned heuristic was chosen for its speed and simplicity, which aided in implementation and rapid experimentation. But as noted in subsection 4.2, other more costly state-of-the-art algorithms, like NSGA-II, may allow still better performance to be gained from learned heuristics. Future work should attempt to use learned heuristics like ours in more varied metaheuristic algorithms.

Another promising place where machine learning can be applied is to learn hyper-heuristics for a metaheuristic algorithm. A typical hyper-heuristic, as used by Ahmed et al. (2019) and Hüsselmann et al. (2023), adapts the probability of using each low-level heuristic based on its performance, while the algorithm is running. It would be interesting to train a neural net policy to act as a hyper-heuristic to select among low-level heuristics, given details about the scenario and previous steps taken during the algorithm.

Our aim in this paper was to show whether learned heuristics can be used in a lightweight metaheuristic algorithm to improve its performance. Our results show that they can, and in fact that they can allow a simple metaheuristic algorithm to set new state-of-the-art results on challenging benchmark cities. This is compelling evidence that they may help transit agencies substantially reduce operating costs while delivering better service to riders in real-world scenarios.

References

Agence métropolitaine de transport, "Enquête origine-destination 2013," 2013, montreal, QC.

L. Ahmed, C. Mumford, and A. Kheiri, "Solving urban transit route design problem using selection hyper-heuristics," *European Journal of Operational Research*, vol. 274, no. 2, pp. 545–559, 2019.

G. Ai, X. Zuo, G. Chen, and B. Wu, "Deep reinforcement learning based dynamic optimization of bus timetable," *Applied Soft Computing*, vol. 131, p. 109752, 2022.

D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "Concorde tsp solver," 2001. [Online]. Available: https://www.math.uwaterloo.ca/tsp/concorde/index.html

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

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.

S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" 2021. [Online]. Available: https://arxiv.org/abs/2105.14491

J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

S. I.-J. Chien, Y. Ding, and C. Wei, "Dynamic bus arrival time prediction with artificial neural networks," *Journal of transportation engineering*, vol. 128, no. 5, pp. 429–438, 2002.

J. Choo, Y.-D. Kwon, J. Kim, J. Jae, A. Hottung, K. Tierney, and Y. Gwon, "Simulation-guided beam search for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8760–8772, 2022.

P. da Costa, J. Rhuggenaath, Y. Zhang, and A. Akcay, "Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Asian conference on machine learning*. PMLR, 2020, pp. 465–480.

H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *arXiv preprint arXiv:1704.01665*, 2017.

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.

M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *CoRR*, vol. abs/1606.09375, 2016. [Online]. Available: http://arxiv.org/abs/1606.09375

J. Durán-Micco and P. Vansteenwegen, "A survey on the transit network design and frequency setting problem," *Public Transport*, vol. 14, no. 1, pp. 155–190, 2022.

D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, 2015.

S. Fortune, "Voronoi diagrams and delaunay triangulations," *Computing in Euclidean geometry*, pp. 225–265, 1995.

Z.-H. Fu, K.-B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large tsp instances," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 8, 2021, pp. 7474–7482.

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

J. Guan, H. Yang, and S. Wirasinghe, "Simultaneous optimization of transit line configuration and passenger line assignment," *Transportation Research Part B: Methodological*, vol. 40, pp. 885–902, 12 2006.

V. Guihaire and J.-K. Hao, "Transit network design and scheduling: A global review," *Transportation Research Part A: Policy and Practice*, vol. 42, no. 10, pp. 1251–1273, 2008.

A. Holliday and G. Dudek, "Augmenting transit network design algorithms with deep learning," in *2023 26th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

——, "A neural-evolutionary algorithm for autonomous transit network design," in *presented at 2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024.

A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," *arXiv preprint arXiv:1911.09539*, 2019.

G. Hüsselmann, J. H. van Vuuren, and S. J. Andersen, "An improved solution methodology for the urban transit routing problem," *Computers & Operations Research*, p. 106481, 2023.

K. A. Islam, I. M. Moosa, J. Mobin, M. A. Nayeem, and M. S. Rahman, "A heuristic aided stochastic beam search algorithm for solving the transit network design problem," *Swarm and Evolutionary Computation*, vol. 46, pp. 154–170, 2019.

R. Jeong and R. Rilett, "Bus arrival time prediction using artificial neural network model," in *Proceedings. The 7th international IEEE conference on intelligent transportation systems (IEEE Cat. No. 04TH8749)*. IEEE, 2004, pp. 988–993.

Z. Jiang, W. Fan, W. Liu, B. Zhu, and J. Gu, "Reinforcement learning approach for coordinated passenger inflow control of urban rail transit in peak hours," *Transportation Research Part C: Emerging Technologies*, vol. 88, pp. 1–16, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X18300111

M. P. John, C. L. Mumford, and R. Lewis, "An improved multi-objective algorithm for the urban transit routing problem," in *Evolutionary Computation in Combinatorial Optimisation*, C. Blum and G. Ochoa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 49–60.

A. Kar, A. L. Carrel, H. J. Miller, and H. T. Le, "Public transit cuts during covid-19 compound social vulnerability in 22 us cities," *Transportation Research Part D: Transport and Environment*, vol. 110, p. 103435, 2022.

K. Kepaptsoglou and M. Karlaftis, "Transit route network design problem: Review," *Journal of Transportation Engineering*, vol. 135, no. 8, pp. 491–505, 2009.

M. Kim, J. Park *et al.*, "Learning collaborative policies to solve np-hard routing problems," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 418–10 430, 2021.

D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

W. Kool, H. V. Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *ICLR*, 2019.

F. Kılıç and M. Gök, "A demand based route generation algorithm for public transit network design," *Computers & Operations Research*, vol. 51, pp. 21–29, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054814001300

C. Li, L. Bai, W. Liu, L. Yao, and S. T. Waller, "Graph neural network for robust public transit demand prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

H. Lin and C. Tang, "Analysis and optimization of urban public transport lines based on multiobjective adaptive particle swarm optimization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 16 786–16 798, 2022.

L. Liu, H. J. Miller, and J. Scheff, "The impacts of covid-19 pandemic on public transit demand in the united states," *Plos one*, vol. 15, no. 11, p. e0242476, 2020.

Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 096–11 107, 2021.

C. E. Mandl, "Evaluation and optimization of urban public transportation networks," *European Journal of Operational Research*, vol. 5, no. 6, pp. 396–404, 1980.

A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

C. L. Mumford, "Supplementary material for: New heuristic and evolutionary operators for the multi-objective urban transit routing problem, cec 2013," https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp.zip, 2013, accessed: 2023-03-24.

——, "New heuristic and evolutionary operators for the multi-objective urban transit routing problem," in *2013 IEEE congress on evolutionary computation*. IEEE, 2013, pp. 939–946.

T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen, "Symbolic regression via neural-guided genetic programming population seeding," *arXiv preprint arXiv:2111.00053*, 2021.

M. Nikolić and D. Teodorović, "Transit network design by bee colony optimization," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5945–5955, 2013.

J.-P. Rodrigue, "Parallel modelling and neural networks: An overview for transportation/land use systems," *Transportation Research Part C: Emerging Technologies*, vol. 5, no. 5, pp. 259–271, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X97000144

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of heuristics*. Springer, 2018, pp. 791–808.

Statistics Canada, "Census profile, 2021 census of population," 2023, accessed: 2023-10-25. [Online].

Available: https://www12.statcan.gc.ca/census-recensement/2021/dp-pd/prof/details/page.cfm

——, "Census dissemination area boundary files," 2021, accessed: 2023-07-01. [Online]. Available: https://www150.statcan.gc.ca/n1/en/catalogue/92-169-X

STL, "Stl gtfs," retreived 2020. [Online]. Available: https://transitfeeds.com/p/societe-de-transport-de-laval/38/1383528159

R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

Q. Sykora, M. Ren, and R. Urtasun, "Multi-agent routing value iteration network," in *International Conference on Machine Learning.* PMLR, 2020, pp. 9300–9310.

R. van Nes, "Multiuser-class urban transit network design," *Transportation Research Record*, vol. 1835, no. 1, pp. 25–33, 2003. [Online]. Available: https://doi.org/10.3141/1835-04

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *arXiv preprint arXiv:1506.03134*, 2015.

T. Wang, Z. Zhu, J. Zhang, J. Tian, and W. Zhang, "A large-scale traffic signal control algorithm based on multi-layer graph deep reinforcement learning," *Transportation Research Part C: Emerging Technologies*, vol. 162, p. 104582, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X24001037

R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, "Learning improvement heuristics for solving routing problems," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 9, pp. 5057–5069, 2021.

Y. Xiong and J. B. Schneider, "Transportation network design using a cumulative genetic algorithm and neural network," *Transportation Research Record*, vol. 1364, 1992.

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, pp. 469–479, 2023.

J. Yang and Y. Jiang, "Application of modified nsga-ii to the transit network design problem," *Journal of Advanced Transportation*, vol. 2020, pp. 1–24, 2020.

H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, "Deepaco: neural-enhanced ant systems for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *CoRR*, vol. abs/1806.01973, 2018. [Online]. Available: http://arxiv.org/abs/1806.01973

S. Yoo, J. B. Lee, and H. Han, "A reinforcement learning approach for bus network design and frequency setting optimisation," *Public Transport*, pp. 1–32, 2023.

L. Zou, J.-m. Xu, and L.-x. Zhu, "Light rail intelligent dispatching system based on reinforcement learning," in *2006 International Conference on Machine Learning and Cybernetics*, 2006, pp. 2493–2496.

M. Y. Çodur and A. Tortum, "An artificial intelligent approach to traffic accident estimation: Model development and application," *Transport*, vol. 24, no. 2, pp. 135–142, 2009.

A Neural Net Architecture

In this appendix, we give a detailed description of the architecture of the neural net used in our GNN policy π_θ.

A.1 Input Features

The policy net operates on three inputs: an $n \times 4$ matrix of node feature vectors X, an $n \times n \times 13$ tensor of edge features E, and a global state vector $\mathbf{s}$.

A node feature x_i is composed of the (x, y) coordinates of the node and its in-degree and out-degree in the street graph.

An edge feature $\mathbf{e}_{ij}$ is composed of the following elements:

- D_{ij}, the demand between the nodes
- $s_{ij} = 1$ if $(i, j, \tau_{ij}) \in \mathcal{E}_s$, 0 otherwise
- τ_{ij} if $s_{ij} = 1$, 0 otherwise
- $c_{ij} = 1$ if $\mathcal{R}$ links i to j, 0 otherwise
- $c_{0ij} = 1$ if j can be reached from i over $\mathcal{R}$ with no transfers, 0 otherwise
- $c_{1ij} = 1$ if one transfer is needed to reach j from i over $\mathcal{R}$, 0 otherwise
- $c_{2ij} = 1$ if two transfers are needed to reach j from i over $\mathcal{R}$, 0 otherwise
- self $= 1$ if $i = j$, 0 otherwise
- $\tau_{\mathcal{R}ij}$ if $c_{ij} = 1$, 0 otherwise
- τ_{rij} if $c_{0ij} = 1$ where r the route that provides the shortest direct trip between i and j, 0 otherwise
- T_{ij}, the shortest-path driving time between the nodes
- α and $1 - \alpha$, the weights of the two components of the cost function C

The global state vector $\mathbf{s}$ is composed of:

- Average passenger trip time given current route set, $C_p(\mathcal{C}, \mathcal{R})$
- Total route time given current route set, $C_o(\mathcal{C}, \mathcal{R})$
- number of routes planned so far, $|\mathcal{R}|$
- number of routes left to plan, $S - |\mathcal{R}|$
- fraction of node pairs with demand $d_{ij} > 0$ that are not connected by $\mathcal{R}$
- α and $1 - \alpha$, the weights of the two components of the cost function C

Note that α and $1 - \alpha$ are included in both $\mathbf{s}$ and $\mathbf{e}_{ij}$.

A.2 Policy Network Architecture

The policy π_θ is a neural net with three components: a GAT "backbone", a halting module NN_{halt}, and an extension module NN_{ext}. The halting and extension modules both operate on the outputs from the GNN. All nonlinearities are ReLU functions. A common embedding dimension of $d_{\text{embed}} = 64$ is used throughout; unless otherwise specified, each component of the system outputs vectors of this dimension.

A.2.1 Backbone network

The backbone is a graph attention net composed of five GATv2(Brody et al., 2021) multi-head graph attention layers separated by ReLU nonlinearities. 4 heads are used in each multi-head attention layer. The network takes as input the node feature collection X and the edge features E, and the final layer outputs a collection of node embeddings $Y = \{\mathbf{y}_i \; \forall i \in \mathcal{N}\}$. These node embeddings are used by the two policy heads to compute action probabilities.

A.2.2 Halting module

NN_{halt} is a simple feed-forward MLP with 2 hidden layers. It takes as input the concatenation of the node embeddings of the first and last nodes on the current route, $\mathbf{s}_t$ and τ_{r_t}. It outputs a scalar h. We apply the sigmoid function to h to get the halting policy:

$$\pi(\text{halt}|s_t) = \sigma(h) = \frac{1}{1+e^{-h}} \tag{14}$$

$$\pi(\text{continue}|s_t) = 1 - \sigma(h) \tag{15}$$

A.2.3 Extension module

The extension module NN_{ext} computes the node-pair scores o_{ij} that are used to calculate the probabilities of selecting each extension on odd-numbered timesteps.

This should be a function of the time between nodes along the path, τ_{aij}, as well as the demand between i and j, and potentially of other features of $\mathcal{C}$ and $G_{\mathcal{R}}$. But since the paths are all shortest paths, $\tau_{aij} = T_{ij}$ in every case, so the score o_{aij} gained by linking nodes i, j is independent of the chosen path a: $o_{aij} = o_{ij}$. Let these be computed by some function $f(\cdot)$:

$$o_{ij} = \begin{cases} f(T_{ij}, i, j, \mathcal{C}, G_{\mathcal{R}}), & \text{if} i \neq j \\ 0, & \text{if} i = j \end{cases} \tag{16}$$

Assuming these node-pair scores are an accurate indication of the benefit of linking those nodes, the score of the path as a whole should simply be the sum of scores of the node-pairs it links:

$$o_a = \sum_{k \in a} \sum_{l \in a} o_{kl} \tag{17}$$

Then, suppose we have a partially-planned route r and are considering paths $a \in \text{SP}$ to extend r. Each of these paths still links all the nodes along it, so can achieve score o_a. But appending a to r will also link every node in r to every node in a, so the benefit of extending r by a should depend on these as well. However, the time taken to get from node $i \in r$ to node $j \in a$ may be greater than T_{ij}, since $r|a$ is not necessarily a shortest path. So the shortest-path node-pair scores o_{ij} here are not appropriate. Instead, new node-pair scores $o_{(r|a)ij}$ should be computed based on the driving time $\tau_{(r|a)ij}$ that would result from joining a to r. Otherwise, the same function $f(\cdot)$ should be appropriate, with the new inter-node time:

$$o_{(r|a)ij} = \begin{cases} f(\tau_{(r|a)ij}, i, j, \mathcal{C}, G_{\mathcal{R}}), & \text{if} i \neq j \\ 0, & \text{if} i = j \end{cases} \tag{18}$$

And the overall score should then be:

$$o_{(r|a)} = o_a + \sum_{i \in r} \sum_{j \in a} \Delta_{i \neq j} o_{(r|a)ij} \tag{19}$$

Where $\Delta_{i \neq j}$ is 0 if $i = j$ and 1 otherwise, and serves to avoid including the score for the overlapping end-node of r and a in the sum.

We design the extension policy head to compute the score function $f(\cdot)$. In this policy head, the function $f(\cdot)$ that computes node-pair scores is an MLP, called MLP_{ext1}, that takes as input T_{ij} or $\tau_{(r|a)ij}$ (as appropriate), the backbone GAT's embeddings y_i and y_j of nodes i and j, and the global state vector $\mathbf{s}$. For each path $a \in \text{SP}$, we sum these node-pair scores to get path scores o_a as in Equation 17, and when extending a route r, we compute the extension scores $o_{(r|a)}$ as in Equation 19. We can efficiently compute the node-pair scores in a batch by pairing all node embeddings y_i when $r_t = []$; and when $r_t \neq []$, we only compute scores $o_{(r|a)ij}$ for node pairs where exactly one of i or j is in r_t. The resultant scalar scores, o_a and $o_{(r|a)}$, are provided to the next stage of the extension policy head.

If $\alpha = 0$, then the benefit of choosing a path a to start or extend a route depends on its driving time τ_a, and generally on the state s. And if $0 < \alpha < 1$, then the benefit of a depends on these as well as on the edges it adds to $\mathcal{E}_{\mathcal{R}}$, and thus on $o_{(r|a)}$. So our architecture ought to incorporate all of these factors. We achieve this with a final MLP, called MLP_{ext2}, that takes as input $\mathbf{s}_t, \tau_a$, and the previous node-pair-based score (o_a or $o_{(r|a)}$, depending on whether $r_t = []$), and outputs a final scalar score $\hat{o}_a$ for each candidate path. The extension policy is then the softmax of these values:

$$\pi_\theta(a|s) = \frac{e^{\hat{o}_a}}{\sum_{a' \in \mathcal{A}} e^{\hat{o}_{a'}}} \tag{20}$$

We thus treat the outputs of NN_{ext} as un-normalized log-probabilities of the possible actions during an extension step.

A.3 Value Network

The value function used to compute the advantage $A_t = G_t^H - V(s_t)$ during training is an MLP with 2 hidden layers of dimension 36. The input to $V(s_t)$ is a vector composed of α and several statistics of $\mathcal{C}$: the average of the node features $\frac{\sum_i \mathbf{x}_i}{n}$, the total demand $\sum_{i,j} D_{ij}$, the means and standard deviations of the elements of D and T, and the cost component weights α and $1 - \alpha$. Along with these, it takes the state vector $\mathbf{s}_t$.

We experimented with using an additional neural net head that would compute the value estimate $V(s_t)$ based on the same node embeddings Y as NN_{ext} and NN_{halt}, but found that this performed slightly worse than using the separate MLP.

Table 7: Training Hyper-Parameters

Hyper-Parameter	Value
Baseline model learning rate	5×10^{-4}
Baseline model weight decay	0.01
Policy learning rate	0.0016
Policy weight decay	8.4×10^{-4}
Number of training iterations	200
Discount rate γ	0.95
PPO CLIP threshold ϵ	0.2
Generalized Advantage Estimation λ	0.95
Batch size	256
Horizon	120
Num. epochs	1
Constraint weight β	5.0
S	10
MIN	2
MAX	12
Adam α	0.001
Adam β_1	0.9
Adam β_2	0.999

B Training Hyper-Parameters

Training was performed using the well-known Adam optimizer Kingma and Ba (2015). Table 7 gives the hyper-parameters used during training. In each step of training, we augment the training dataset by applying a set of random transformations. These include rescaling the node positions by a random factor uniformly sampled in the range $[0.4, 1.6]$, rescaling the demand magnitudes by a random factor uniformly sampled in the range $[0.8, 1.2]$, mirroring node positions about the y axis with probability 0.5, and revolving the node positions by a random angle in $[0, 2\pi)$ about their geometric center.

We also randomly sample α separately for each training city in each batch. Each time alpha is sampled, it has equal probability of being set to 0.0, set to 1.0, and sampled uniformly in the range $[0, 1]$. This is to encourage the policy to learn to handle not only intermediate cases, but also the extreme cases of the passenger perspective and operator perspective.

This data augmentation lends greater variety to the finite dataset, increasing the performance of the resulting policy on new environments not seen during training.

It is common to incorporate an entropy term in the loss function when training policy gradient models. However, we found that doing so harmed performance of the resulting models on this problem, so we did not use such a term when training the model with PPO.

A set of preliminary experiments with a range of values for the discount rate parameter γ led us to set this parameter to $\gamma = 0.95$.