

Learning heuristics for transit network design and improvement with deep reinforcement learning

Andrew Holliday, Ahmed El-Geneidy & Gregory Dudek

To cite this article: Andrew Holliday, Ahmed El-Geneidy & Gregory Dudek (2025) Learning heuristics for transit network design and improvement with deep reinforcement learning, *Transportmetrica B: Transport Dynamics*, 13:1, 2561863, DOI: [10.1080/21680566.2025.2561863](https://doi.org/10.1080/21680566.2025.2561863)

To link to this article: <https://doi.org/10.1080/21680566.2025.2561863>



© 2025 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.



Published online: 17 Oct 2025.



[Submit your article to this journal](#)



Article views: 77




[View related articles](#)



[View Crossmark data](#)

Learning heuristics for transit network design and improvement with deep reinforcement learning

Andrew Holliday ^a, Ahmed El-Geneidy^b and Gregory Dudek^a

^aMcGill University Center for Intelligent Machines, Montreal, Canada; ^bMcGill University School of Urban Planning, Montreal, Canada

ABSTRACT

Planning a network of public transit routes is a challenging optimisation problem. Meta-heuristic algorithms search through the space of possible transit networks by applying heuristics that randomly alter routes in a network. Existing algorithms almost exclusively use heuristics that modify the network in purely random ways. In this work, we explore whether we can obtain better transit networks using more intelligent heuristics, that modify networks according to a learned preference function instead of at random. We use reinforcement learning to train graph neural nets to act as heuristics. These neural heuristics yield improved results on benchmark synthetic cities with 70 nodes or more, and achieve new state-of-the-art results on the challenging Mumford benchmark. They also improve upon a simulation of the real transit network in the city of Laval, Canada, achieving cost savings of up to 19% over the city's existing transit network.

ARTICLE HISTORY

Received 28 February 2025
Accepted 13 July 2025

KEYWORDS

Public transit; urban planning; neural networks; reinforcement learning; optimisation

1. Introduction

The COVID-19 pandemic caused declines in transit ridership in cities around the world (Liu, Miller, and Scheff 2020), putting many transit agencies under pressure to reduce operating costs (Kar et al. 2022). By improving a transit network's spatial layout, it may be possible to reduce costs while improving service quality. But network redesigns can be very costly and are disruptive for riders, so it is vital that agencies get a design (or re-design) 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 high-quality set of transit routes for a city. It is an NP-hard problem, and real-world cities typically have hundreds or even thousands of transit stops, making analytical optimisation approaches infeasible. The most successful approaches to the NDP to-date have been metaheuristic algorithms. These repeatedly apply heuristics that randomly change a network, and use a metaheuristic rule such as natural selection or metallic annealing to decide which changes to keep.

Different heuristics used in these algorithms make different kinds of changes to networks, such as randomly removing a stop from a route, or randomly exchanging two stops on a route. What most heuristics have in common is that they make changes purely at random, ignoring the particulars of the network or city. In this work, we use deep reinforcement learning (DRL) to train neural-net-based heuristics to choose changes to a transit network based on information about the city and the current network. We integrate these with a simple evolutionary algorithm to form what we call a Neural Evolutionary Algorithm, and we show that in challenging scenarios, this technique outperforms existing non-neural algorithms.

This paper expands on our prior work (Holliday and Dudek 2023, 2024) in a number of ways. We present new results obtained with an improved evolutionary algorithm and an improved graph neural net (GNN) model trained via Proximal Policy Optimization instead of REINFORCE. We compare these results with state-of-the-art methods on the widely-used (Mandl 1980) and (Mumford 2013a) benchmark cities, and find that on the most challenging benchmarks our method sets a new state of the art, improving on other methods by up to 4.8%.

We also report on ablation studies that show the importance of different features of our method. As part of these studies, we consider a novel non-neural heuristic, where shortest paths with common end-points are

CONTACT Andrew Holliday  andrew.holliday@mail.mcgill.ca  McGill University Center for Intelligent Machines, 3480 University Street, McConnell Engineering Building, Room 410, Montreal, QC H3A 0E9, Canada

© 2025 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

This is an Open Access article distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited, and is not altered, transformed, or built upon in any way. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

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 minimising 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 perform a case study, applying our neural heuristics to a simulation of the real-world city of Laval, Canada, showing that they can plan transit networks that improve on the city's existing transit system by a wide margin by three distinct optimisation criteria. These results show that neural heuristics may allow transit agencies to offer better service at less cost.

2. Background and related work

2.1. Deep learning for optimisation problems

Deep reinforcement learning refers to the use of 'deep' artificial neural nets – that is, neural nets with many hidden layers – as reinforcement learning (RL) models. RL is a branch of machine learning concerned with learning from feedback that comes in the form of a numerical 'reward' that indicates how good or bad the model's outputs are, so that its outputs change to maximise the reward it receives. As documented in Bengio, Lodi, and Prouvost (2021), there is growing interest in the application of deep RL to combinatorial optimisation (CO) problems.

Vinyals, Fortunato, and Jaitly (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). This work has been built on by Dai et al. (2017), Kool, Hoof, and Welling (2019), Sykora, Ren, and Urtasun (2020) and others, training similar neural net models with reinforcement learning 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 a 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, Qiu, and Zha (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, Hoof, and Welling (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 specialised algorithms such as Concorde (Applegate et al. 2001), while requiring much less run-time to compute (Kool, Hoof, and Welling 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. Because this search process can continue indefinitely, improvement methods are generally more computationally costly than construction methods, but can yield better results. Evolutionary algorithms belong to this category.

Some work (Chen and Tian 2019; da Costa et al. 2020; Hottung and Tierney 2019; Ma et al. 2021; Y. Wu 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, Park, and Kim (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 optimisation 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; Defferrard, Bresson, and Vandergheynst 2016; Duvenaud et al. 2015; Kipf and Welling 2016). 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;

Table 1. Statistics of the Mandl and Mumford benchmark cities.

City	# nodes n	# link edges $ \mathcal{E}_s $	# 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

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, Lodi, and Prouvost (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. For example, take the 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 $\binom{2.3 \times 10^{11}}{12} \approx 4.3 \times 10^{127}$ 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. These differences necessitate a special treatment, so our work deals only with the NDP, as does most other recent work on the problem (Ahmed, Mumford, and Kheiri 2019; Hüselmann, van Vuuren, and Andersen 2024; Islam et al. 2019; John, Mumford, and Lewis 2014; Kılıç and Gök 2014; Lin and Tang 2022; Mumford 2013a; Zervas et al. 2024).

While analytical and mathematical programming methods have been successful on small instances (Guan, Yang, and Wirasinghe 2006; van Nes 2003), they struggle to realistically represent the problem (Guihaire and Hao 2008; Kepaptsoglou and Karlaftis 2009). Metaheuristic approaches, as defined by Sörensen, Sevaux, and Glover (2018), have thus been more widely applied, both for the NDP and the related Frequency-Setting Problem (Aksoy and Mutlu 2024).

The most widely-used metaheuristics for the NDP have been genetic algorithms, simulated annealing, and ant-colony optimisation, along with hybrids of these methods (Durán-Micco and Vansteenwegen 2022; Guihaire and Hao 2008; Hüselmann, van Vuuren, and Andersen 2024; Kepaptsoglou and Karlaftis 2009; Yang and Jiang 2020). Recent work has also shown other metaheuristics can be used with success, such as sequence-based selection hyper-heuristics (Ahmed, Mumford, and Kheiri 2019), beam search (Islam et al. 2019), and particle swarms (Lin and Tang 2022). Many different heuristics have been applied within these metaheuristic algorithms, but most have in common that they select among possible neighbourhood moves uniformly at random.

An exception is Hüselmann, van Vuuren, and Andersen (2024). For two heuristics, the authors design a simple model of how each change the heuristic could make would affect the network's quality. The heuristics then weight the probability of making different changes according to this model, with higher-quality changes being more likely. 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 been used for predictive problems in urban mobility (Chien, Ding, and Wei 2002; Çodur and Tortum 2009; Jeong and Rilett 2004; Li, Bai, Liu et al. 2020; Rodrigue 1997; F. Wu et al. 2025; Xiong and Schneider 1992) and for other transit optimisation problems such as scheduling, passenger flow control, and traffic signal control (Ai et al. 2022; Jiang et al. 2018; Wang et al. 2024; F. Wu et al. 2025; Yan et al. 2023;

Zhang et al. (2023; Zou, Xu, and Zhu 2006), they have not often been applied to the NDP. The same is true of RL: Li, Bai, Yao et al. (2023) review the work on the application of RL to public transit, and find only one instance in which it was used for network design: the work of Wei et al. (2020). In this work, the authors train a neural net policy via RL to design a single new metro route for the city of Xi'an, China. They obtain good results in that case, but their model of the problem differs from ours in not requiring the network to connect all stations, and their method does not go beyond planning a single route.

Two other recent examples are Darwish, Khalil, and Badawi (2020) and Yoo, Lee, and Han (2023). Both use RL to design routes and a schedule for the Mandl benchmark (Mandl 1980), a very small city, and both obtain good results. Darwish, Khalil, and Badawi (2020) use a GNN approach inspired by Kool, Hoof, and Welling (2019); in our own work we experimented with a nearly identical approach to Darwish, Khalil, and Badawi (2020), but we found it did not scale beyond very small instances. Meanwhile, Yoo, Lee, and Han (2023) use tabular RL, an approach that is practical only for small problem sizes.

All three of these approaches also require a new model to be trained on each problem instance. Our approach, by contrast, finds good solutions for NDP instances of more than 600 nodes, and can be applied to instances unseen during model training.

3. The transit network design problem

In the transit network design problem, a city is represented as an augmented graph:

$$\mathcal{G} = (\mathcal{N}, \mathcal{E}_s, D) \quad (1)$$

This consists of a set \mathcal{N} of n nodes, representing candidate stop locations; a set \mathcal{E}_s of link edges (i, j, τ_{ij}) representing streets or railways that connect the nodes, with weights τ_{ij} indicating travel times on those links; and an $n \times n$ matrix D giving the travel demand, in number of trips, between every pair of nodes in \mathcal{N} . Our goal is to find a transit network \mathcal{R} that minimises a cost function $C : \mathcal{G}, \mathcal{R} \rightarrow \mathbb{R}^+$. \mathcal{R} is a set of routes where each route $r \in \mathcal{R}$ is a sequence of nodes in \mathcal{N} . \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 given at the start.
- (3) All routes $r \in \mathcal{R}$ must obey $MIN \leq |r| \leq MAX$, where MIN and MAX are parameters set by the user.
- (4) Routes $r \in \mathcal{R}$ may not contain cycles; each node $i \in \mathcal{N}$ can appear in r at most once.

An example city graph with a transit network is shown in Figure 1.

We deal here with the symmetric NDP, meaning that all demand, links, and routes are the same in both directions:

$$D = D^T \quad (2)$$

$$(i, j, \tau_{ij}) \in \mathcal{E}_s \quad \text{iff.} \quad (j, i, \tau_{ji}) \in \mathcal{E}_s \quad (3)$$

and all routes are traversed both forwards and backwards by vehicles on them. This allows us to evaluate our method against the large number of other methods that evaluate on symmetric benchmark cities, but we note that extending our method to the asymmetric NDP would be straightforward.

3.1. Cost functions

Work on the NDP usually considers two separate cost functions: the passenger cost C_p , and the operating cost C_o (Ahmed, Mumford, and Kheiri 2019; Hüsselman, van Vuuren, and Andersen 2024; John, Mumford, and Lewis 2014; Kılıç and Gök 2014; Nikolić and Teodorović 2013). We present the standard definitions of these terms used in this work below.

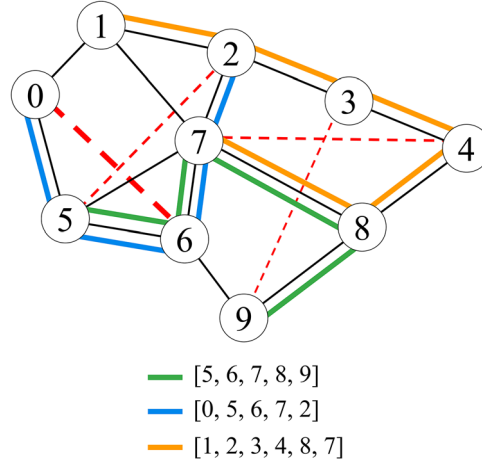


Figure 1. An example city graph with ten numbered nodes and three routes. Link edges (representing streets or railways) are black, routes are in colour, and demands are shown by dashed red lines. The edges of the three routes form a subgraph of the link graph $(\mathcal{N}, \mathcal{E}_s)$. All node-pairs 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. The demand from 3 to 9 requires passengers to ride the orange line from node 3 to 8, and transfer to the green line to go from 8 to 9.

C_p is defined as the average time of passenger trips over the network:

$$C_p(\mathcal{G}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij} \tau_{\mathcal{R}ij}}{\sum_{i,j} D_{ij}} \quad (4)$$

where $\tau_{\mathcal{R}ij}$ is the time of the shortest transit trip from i to j given \mathcal{R} , including a time penalty for each transfer between routes.

The operating cost is the total driving time of the routes:

$$C_o(\mathcal{G}, \mathcal{R}) = \sum_{r \in \mathcal{R}} \tau_r \quad (5)$$

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

We use these standard definitions in evaluating our algorithm. However, our metaheuristic detailed in Subsection 4.2 requires a single cost function to operate, and our DRL technique requires that this function not range much beyond $[-1, 1]$, as numerically large values can cause problems with neural net training. So we define a unified cost function C that combines passenger and operating costs and a third term that penalises constraint violations that may occur during training.

In C , we use a modified passenger cost C'_p :

$$C'_p(\mathcal{G}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij} (\delta_{\mathcal{R}ij} \tau_{\mathcal{R}ij} + (1 - \delta_{\mathcal{R}ij}) 2 \max_{k,l} T_{kl})}{\sum_{i,j} D_{ij}} \quad (6)$$

$\delta_{\mathcal{R}ij}$ is a delta function with value 1 if \mathcal{R} provides a path from i to j , and 0 if not. T is an $n \times n$ matrix of shortest-path travel times between every node pair. Each demand that is not satisfied induces a penalty of $2 \max_{k,l} T_{kl}$, penalising it, where $\max_{k,l} T_{kl}$ is the maximum travel time between any two nodes in the graph. If \mathcal{R} satisfies constraint 1, then $\delta_{\mathcal{R}ij} = 1 \forall i, j$ and C'_p reduces to the original C_p .

The constraint-violation cost C_c is:

$$C_c(\mathcal{G}, \mathcal{R}) = F_{un} + F_s + 0.1 \delta_v \quad (7)$$

F_{un} is the fraction of node pairs (i, j) with $D_{ij} > 0$ for which \mathcal{R} provides no path. F_s is a smoothly increasing function of the extent to which the routes in \mathcal{R} violate constraint 3:

$$F_s = \frac{\sum_{r \in \mathcal{R}} \max(0, MIN - |r|, |r| - MAX)}{S * MAX} \quad (8)$$

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

This definition implies that when there are no constraint violations, $C_c = 0$. We use fractional measures F_{un} and F_s so that usually, $C_c < 2$, as long as MIN and MAX scale roughly with n . However, it has the drawback that if n is large, 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 some constraint violations being permitted. To prevent this, we include the term $0.1\delta_v$, which ensures that even one violation will significantly increase overall cost $C(\mathcal{G}, \mathcal{R})$, no matter the size of the city graph.

C_c does not penalise violations of constraints 2 and 4, because the Markov Decision Process (MDP) detailed in Subsection 3.2 and the metaheuristic detailed in Subsection 4.2 are, by design, incapable of producing networks that violate these constraints.

The unified cost function is:

$$C(\mathcal{G}, \mathcal{R}) = \alpha w_p C'_p + (1 - \alpha) w_o C_o + \beta C_c \quad (9)$$

The weight $\alpha \in [0, 1]$ controls the trade-off between passenger and operating costs, while β is the constraint violation weight. w_p and w_o are re-scaling constants chosen so that $w_p C'_p$ and $w_o C_o$ both vary roughly over the range $[0, 1]$ for different \mathcal{G} and \mathcal{R} . The values used are:

$$w_p = \left(\max_{ij} T_{ij} \right)^{-1} \quad (10)$$

$$w_o = \left(S \max_{ij} T_{ij} \right)^{-1} \quad (11)$$

3.2. Markov decision process formulation

A Markov Decision Process (MDP) is a formalism 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 . At each timestep t , the environment is in a state s_t , 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} according to the state transition distribution $P(s_{t+1} | s_t, a_t)$, and the agent receives a reward $R_t \in \mathbb{R}$ according to the reward function $R_t = f_R(s_t, a_t, s_{t+1})$. 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 to find a policy π that maximises the return G_t , defined as the time-discounted sum of rewards:

$$G_t = \sum_{t'=t}^{t_{\text{end}}} \gamma^{t'-t} R_{t'} \quad (12)$$

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

We here define an MDP that constructs a transit network for the transit network design problem (Figure 2). As shown in Equation (13), 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:

$$s_t = (\mathcal{R}_t, r_t) \quad (13)$$

The starting state s_1 is $(\mathcal{R}_1 = \{\}, r_1 = \square)$. 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{G} . If $r_t = \square$, then:

$$\mathcal{A}_t = \begin{cases} \{a \mid a \in \text{SP}, |a| \leq \text{MAX}\} & \text{if } r_t = \square \\ \text{EX}_{r_t} & \text{otherwise} \end{cases} \quad (14)$$

where EX_{r_t} is the set of paths $a \in \text{SP}$ that satisfy all of the following conditions:

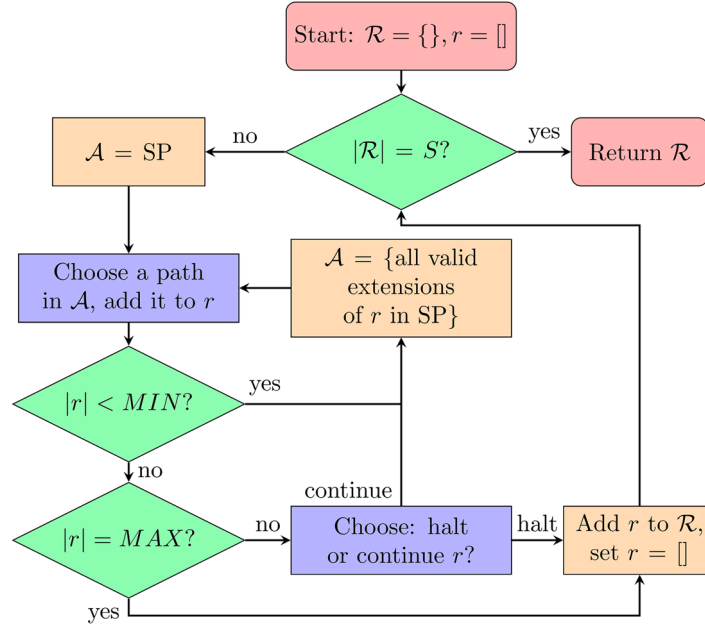


Figure 2. A flowchart of the transit network construction process defined by our MDP. Blue boxes indicate points where the timestep t is incremented and the agent selects an action. Red nodes are the beginning and ending of the process. Green nodes are hard-coded decision points. Orange nodes show updates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

- $(i, j, \tau_{ij}) \in \mathcal{E}_s$, 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 :

$$\mathcal{A}_t = \begin{cases} \{\text{continue}\} & \text{if } |r_t| < MIN \\ \{\text{halt}\} & \text{if } |r_t| = MAX \text{ or } |EX_{r_t}| = 0 \\ \{\text{continue, halt}\} & \text{otherwise} \end{cases} \quad (15)$$

And the route set \mathcal{R}_{t+1} and current route r_{t+1} are determined by the chosen action as follows:

$$(\mathcal{R}_{t+1}, r_{t+1}) = \begin{cases} (\mathcal{R}_t \cup \{r_t\}, []) & \text{if } a_t = \text{halt} \\ (\mathcal{R}_t, r_t) & \text{if } a_t = \text{continue} \end{cases} \quad (16)$$

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 , and \mathcal{R}_{t+1} is the final transit network. The reward function is defined as the decrease in cost between each consecutive pair of timesteps:

$$R_t = C'(\mathcal{G}, \mathcal{R}_t \cup \{r_t\}) - C'(\mathcal{G}, \mathcal{R}_{t+1} \cup \{r_{t+1}\}) \quad (17)$$

where $C'(\mathcal{G}, \mathcal{R}) = C(\mathcal{G}, \mathcal{R}) - \beta F_s$; that is, C' is the cost function without the F_s term that penalises violations of constraint 3. We remove F_s because the MDP structure guarantees that the resulting networks satisfy constraint 3.

This MDP formalisation 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 directness by forcing them to be composed of shortest paths. While an agent may construct indirect routes by choosing paths with length 2 at every step, this is unlikely because in realistic link graphs, the majority of paths in SP are longer than two nodes. Third, the alternation between deciding to continue or halt r_t and deciding how to extend r_t means that the probability of

halting does not depend on how many different extensions are available; so a policy learned in environments with few possible extensions should generalise to environments with many, and vice versa.

4. Neural heuristics

4.1. Learning to construct a network

To build a neural heuristic, we first train a GNN policy $\pi_\theta(a | s)$, parameterised by θ , to maximise the cumulative return G_t on the construction MDP described in Subsection 3.2. By following this learned policy on the MDP for some city \mathcal{G} , stochastically sampling a_t from $\pi(\cdot | s_t)$ at each timestep, we obtain a transit network \mathcal{R} for that city. We refer to this learned-policy-based construction algorithm as ‘learned construction’. Because learned construction samples actions stochastically, we can run it repeatedly to generate multiple networks for a city. ‘LC-100’ denotes the procedure whereby we run learned construction 100 times and choose the best of the resulting networks. As noted in Subsection 2.2, the NDP has a much wider space of possible solutions than problems such as the TSP or VRP. To make this tractable, we found it necessary to ‘factorise’ the computation of the policy on odd-numbered timesteps, since \mathcal{A}_t may be very large. Our policy net outputs a scalar score o_{aij} for each node pair (i, j) in the graph and each possible extension a . From these, a score o_a for each extension is computed by summing over o_{aij} for all (i, j) that would become directly connected by appending path a to the current route r_t :

$$o_a = \sum_{i \in a, j \in a, i \neq j} o_{aij} + \sum_{i \in r_t, j \in a} o_{aij} \quad (18)$$

where $o_{aij} = 0$ for all i . With this factorisation, the learning objective becomes estimating the benefit of directly connecting nodes i and j with travel time $\tau_{(r_t | a)ij}$. This is an easier objective than directly estimating the merit of adding path a to r_t .

The central component of the policy net is a graph attention net (GAT) (Brody, Alon, and Yahav 2021) which treats the city \mathcal{G} as a fully-connected graph with node set \mathcal{N} . Each node i has a feature vector \mathbf{x}_i , and each pair of nodes (i, j) also has a feature vector \mathbf{e}_{ij} . These contain information about location, demand, existing transit routes, and the link edge (if one exists) between i and j .

The GAT takes the sets of node and node-pair feature vectors, and outputs a set of node embeddings $Y = \{\mathbf{y}_i \mid i \in \mathcal{N}\}$. These are used as input to one of two ‘head’ neural nets, NN_{ext} or NN_{halt} , depending on the timestep. When the timestep t is odd, NN_{ext} takes Y and the driving time from i to j along $r_t | a$ (the concatenation of r_t with a), and computes the node-pair scores $o_{ij} \forall i, j \in \mathcal{N} \times \mathcal{N}$. When t is even, NN_{halt} takes Y and other statistics of \mathcal{G} and $\mathcal{R}_t \cup \{r_t\}$ and outputs a probability of choosing ‘halt’ vs ‘continue’. The details of these components are presented in Appendix 1, along with the details of the node and edge feature vectors. We have released the code for our method and experiments to the public.¹

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), with a reward function that gave a reward only at the end of the MDP.:

$$R_t = \begin{cases} -C(\mathcal{G}, \mathcal{R}_t) & \text{if } t = t_{\text{end}} \\ 0 & \text{if } t < t_{\text{end}} \end{cases} \quad (19)$$

This was inspired by the similar approach taken by Kool, Hoof, and Welling (2019). However, sparse reward functions like this one, where $R_t = 0$ at most timesteps, convey little information about the effect of each action, making learning more difficult. So we here use the richer reward function we describe in Section 3.2.

REINFORCE has also been superseded by more recent policy gradient learning algorithms. So in this work we train the policy net using Proximal Policy Optimization with Generalized Advantage Estimation, a policy gradient method proposed by Schulman et al. (2017). Proximal Policy Optimization (PPO) was chosen for its relative simplicity and for the fact that it still achieves state-of-the-art performance on many problems. PPO works by running H timesteps of an MDP episode (or multiple episodes in parallel), and then computing an

‘advantage’ A_t for each timestep:

$$A_t = \sum_{t'=0}^{H-1} \gamma^{t'} R_{t+t'} + \gamma^H V(s_{t+H}) - V(s_t) \quad (20)$$

From this we compute the generalised advantage, \hat{A}_t , a weighted sum of A_t over a range of values of the horizon, H . \hat{A}_t weights the updates to the policy, such that actions with a positive \hat{A}_t are made more likely, and those with a negative \hat{A}_t are made less so.

The value function $V(s_t)$ is itself a small Multi-Layer Perceptron (MLP) neural net that is trained in parallel with the policy net. It takes the the cost weight parameter α and statistics of the city \mathcal{G} and of the current network $\mathcal{R}_t \cup \{r_t\}$ as input, and is trained to predict the discounted return G_t^H :

$$G_t^H = \sum_{t'=0}^{H-1} \gamma^{t'} R_{t+t'} + \gamma^H V(s_{t+H}) \quad (21)$$

We train the policy and value nets on a dataset of synthetic cities. At each iteration, a full MDP episode is run in parallel on each city in a ‘batch’ of cities from the dataset. We sample a different $\alpha \sim [0, 1]$ for each training MDP episode, while holding S , MIN , MAX , and the constraint weight β constant. The values to which we set these hyperparameters are presented in Table A1. $C(\mathcal{G}, \mathcal{R}_t \cup \{r_t\})$, $V(s_t)$ and \hat{A}_t are computed for each timestep of each episode in the batch, and the policy and value nets are updated accordingly via PPO.

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

- 4-nn: Create the node set \mathcal{N} by sampling n 2D points uniformly at random over a square. Then create \mathcal{E}_s by making link edges between each node i and its four nearest neighbours.
- 4-grid: Place n nodes in a rectangular grid as close to square as possible. Create \mathcal{E}_s by making link edges between all horizontal and vertical neighbours.
- 8-grid: The same as 4-grid, but also make link edges between diagonal neighbours.
- Voronoi: Sample m random 2D points in a square, and compute their Voronoi diagram (Fortune 1995). Let \mathcal{N} and \mathcal{E}_s be the shared vertices and edges of the resulting Voronoi cells. m is chosen so $|\mathcal{N}| = n$.

In each process, we sample nodes (or the m points for Voronoi) in a 30 km \times 30 km square, and assume a fixed vehicle speed of $v = 15$ m/s to calculate link edge weights $\tau_{ij} = \|(x_i, y_i) - (x_j, y_j)\|_2 / v$. After each process except Voronoi, we delete each edge in \mathcal{E}_s with probability ρ : if the resulting graph $\{\mathcal{N}, \mathcal{E}_s\}$ is not connected, we discard it and repeat the process. This edge-deletion step adds more variety to the layouts of each type of city, especially 4-grid and 8-grid types, which would otherwise all have the same structure.

Finally, we generate the demand 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 $D_{ji} = D_{ij}$ if $i < j$ to make D symmetric.

We generate a dataset of $2^{15} = 32,768$ of these synthetic cities, setting $n = 20$ and $\rho = 0.3$. During training, we randomly choose 90% of these cities to form the training set, and set aside the remaining 10% as a validation set.

In each iteration of training, we train on a batch of cities from the training set. After every ten iterations, we run learned construction with the current policy net on every city in the validation set, and record the average cost C_{val} of the resulting networks. At the end of training, we return the parameters θ that achieved the lowest C_{val} , 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.

During training, all neural net inputs are normalised so as to have zero mean and unit variance across the entire dataset. The scaling and shifting 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

To test the usefulness of a neural net policy for metaheuristic algorithms, we devised a ‘neural heuristic’ that makes use of such a policy, and compared a simple baseline evolutionary algorithm to a variant of the baseline that uses this neural heuristic. In this section, we describe these two evolutionary algorithms, as well as our neural heuristic.

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 of its good performance for relatively little computational cost, but we note that our learned policies could be used as heuristics in a wide variety of metaheuristic algorithms. Nikolić and Teodorović (2013) describe their algorithm as a ‘bee colony optimisation’ algorithm. As noted in Sörensen (2015), ‘bee colony optimisation’ is mathematically identical to one kind of evolutionary algorithm, an older and well-established class of metaheuristic algorithms. To avoid confusion and the spread of unnecessary terminology, we here refer to as an evolutionary algorithm and avoid the ‘bee colony’ terminology.

The algorithm operates on a population of B solutions $\mathcal{B} = \{\mathcal{R}_b \mid 1 \leq b \leq B\}$, and performs alternating stages of mutation and selection. In the mutation stage, the algorithm applies two heuristics, type 1 and type 2, to different subsets of the population chosen at random. If the modified 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 $C(\mathcal{G}, \mathcal{R}_b)$. After IT repetitions of mutation and selection, the algorithm returns the network \mathcal{R} with the lowest $C(\mathcal{G}, \mathcal{R})$ from all iterations.

The type-1 and type-2 heuristics both begin by selecting, uniformly at random, a route r in \mathcal{R}_b and a terminal node i on r . The type-1 heuristic then selects a random node $j \neq i$ in \mathcal{N} , and replaces r with SP_{ij} , the shortest path between i and j . The probability of choosing each node j is proportional to the amount of demand directly satisfied by SP_{ij} . The type-2 heuristic does one of two things: with probability p_d , it deletes i from r ; otherwise, it adds a random node j in i ’s link-graph neighbourhood to r , making j the new terminal. Following Nikolić and Teodorović (2013), we set $p_d = 0.2$ in our experiments.

The initial population of solutions is obtained by making B copies of a single initial solution, \mathcal{R}_0 . In our prior work (Holliday and Dudek 2023), we found that using LC-100 to obtain \mathcal{R}_0 outperformed Nikolić and Teodorović (2013)’s own method, so that is what both baseline and variant evolutionary algorithms do here. We refer to this baseline evolutionary algorithm as ‘EA’.

Our variant algorithm differs from EA only in that instead of using the type-1 heuristic, it uses a ‘neural heuristic’ which is based on our neural net policy. The neural heuristic selects a route $r \in \mathcal{R}_b$ at random, removes it from the network to get $\hat{\mathcal{R}}_b = \mathcal{R}_b \setminus r$, and then runs learned construction with π_θ starting from $\hat{\mathcal{R}}_b$. Because $|\hat{\mathcal{R}}_b| = S - 1$, this produces just one new route, r' , which is then added to the network to get $\mathcal{R}'_b \leftarrow \hat{\mathcal{R}}_b \cup \{r'\}$. In this way we leverage π_θ , which was trained as a policy for a construction method, to aid in an improvement method. We refer to this variant algorithm as the neural evolutionary algorithm (NEA). Both algorithms are presented in Algorithm 1, with the difference between the two described in lines 10 and 11.

We replace the type-1 heuristic with the neural heuristic because they each choose from a similar space of changes: replacing one route by a shortest path in the former case, and replacing one route by a new route composed of shortest paths in the latter.

5. Benchmark experiments

We first evaluated our method on the Mandl (1980) and Mumford (2013b) city datasets, two popular benchmarks for evaluating NDP algorithms (Ahmed, Mumford, and Kheiri 2019; John, Mumford, and Lewis 2014; Kılıç and Gök 2014; Mumford 2013a). Mandl is just one small synthetic city with fifteen nodes. The Mumford dataset consists of four synthetic cities, labelled Mumford0 through Mumford3, and it specifies 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. More details of these benchmark cities are given in Table 1.

Algorithm 1 Evolutionary Algorithm (Baseline and Neural)

```

1: Input:  $\mathcal{G} = (\mathcal{N}, \mathcal{E}_s, D), SP, C, B, IT, E$ 
2: Construct initial network  $\mathcal{R}_0 \leftarrow \text{LC-100}(\mathcal{G}, C)$ 
3:  $\mathcal{R}_b \leftarrow \mathcal{R}_0 \forall b \in \text{integers } 1 \text{ through } B$ 
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:        if baseline algorithm (EA) then:  $\mathcal{R}'_b \leftarrow \text{type\_1\_heuristic}(\mathcal{R}_b)$ 
11:        if neural variant (NEA) then:  $\mathcal{R}'_b \leftarrow \text{neural\_heuristic}(\mathcal{R}_b)$ 
12:      else
13:         $\mathcal{R}'_b \leftarrow \text{type\_2\_heuristic}(\mathcal{R}_b)$ 
14:      if  $C(\mathcal{G}, \mathcal{R}'_b) < C(\mathcal{G}, \mathcal{R}_b)$  then
15:         $\mathcal{R}_b \leftarrow \mathcal{R}'_b$ 
16:      Randomly shuffle network indices  $b$ 
17:   // Selection stage
18:    $C_{\text{max}} \leftarrow \max_b C(\mathcal{G}, \mathcal{R}_b)$ 
19:    $C_{\text{min}} \leftarrow \min_b C(\mathcal{G}, \mathcal{R}_b)$ 
20:   for  $b = 1$  to  $B$  do
21:     if  $C(\mathcal{G}, \mathcal{R}_b) < C(\mathcal{G}, \mathcal{R}_{\text{best}})$  then
22:        $\mathcal{R}_{\text{best}} \leftarrow \mathcal{R}_b$ 
23:   // Select surviving networks
24:    $O_b \leftarrow \frac{C_{\text{max}} - C_b}{C_{\text{max}} - C_{\text{min}}}$ 
25:    $s_b \sim \text{Bernoulli}(1 - e^{-O_b})$ 
26:   // Set survivor reproduction probabilities
27:    $p_b \leftarrow \frac{O_b s_b}{\sum_{b'} O_{b'} s_{b'}}$ 
28:   if  $\exists b \in [1, B]$  s.t.  $s_b = 1$  then
29:     // Replace non-survivors with survivors' offspring
30:     for  $b = 1$  to  $B$  do
31:       if  $s_b = 0$  then
32:          $k \sim P(K)$ , where  $P(K = b') = p_{b'}$ 
33:          $\mathcal{R}_b \leftarrow \mathcal{R}_k$ 
34: return  $\mathcal{R}_{\text{best}}$ 

```

In all of our experiments, we set the transfer time penalty used in computing the average trip time C_p to 300s (five minutes). This value is widely used in other work that reports results on these MDP benchmarks (Mumford 2013a), so using this value of the transfer time penalty allows us to directly compare our own results with these other results.

5.1. Comparison with baseline evolutionary algorithm

We ran 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 let us observe how well the different methods perform over a range of possible user preferences, including 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) as well as a range of intermediate perspectives. In all of these experiments, we set the constraint weight β to 5.0, the same value used in training the policies. We found that this was sufficient to prevent any of the constraints listed in Section 3 from being violated by any transit network produced in our experiments.

Table 2. Cost $C(\alpha, \mathcal{G}, \mathcal{R})$ achieved by LC-100, EA, and NEA for three different α values. Costs are averaged over ten random seeds; the \pm value is the standard deviation of $C(\alpha, \mathcal{G}, \mathcal{R})$ 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

Because these are stochastic algorithms, for each benchmark city we ran each algorithm ten times with ten different random seeds. For algorithms that make use of a learned policy, we trained ten separate policies with the same set of random seeds (but using the same training dataset), and used each policy when running algorithms with the corresponding random seed. The values we report are statistics averaged over these ten runs.

We first compared our neural evolutionary algorithm (NEA) with our baseline evolutionary algorithm (EA). We also compared 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 $C(\mathcal{G}, \mathcal{R})$ 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. We observe that there is a trade-off between C_p and C_o : C_o increases and C_p decreases as α increases. We also observe that for $\alpha < 1.0$, NEA pushes C_o much lower than LC-100 and EA. Importantly, for any network \mathcal{R} from LC-100 or EA, there is some value of α for which NEA produces a network \mathcal{R}' that strictly dominates \mathcal{R} , having equal or lower C_p and C_o . Figure 3 also shows results for an additional algorithm, LC-40k, which we discuss in Subsection 5.2.

On Mumford1, Mumford2, and 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. We also observe that LC-100 favours reducing C_p over C_o , with its points clustered at higher C_o and lower C_p than the points with corresponding α values on the other curves. EA and NEA each 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 largest Mumford cities (1, 2, and 3), and over the same range of α values as in Subsection 5.1.

5.2.1. Effect of number of samples

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. To test how much of NEA’s superiority to LC-100 is due to this difference, we repeated our experiments with the LC-40k algorithm, which is the same as LC-100 except that it samples 40,000 networks from the learned-construction procedure instead of 100.

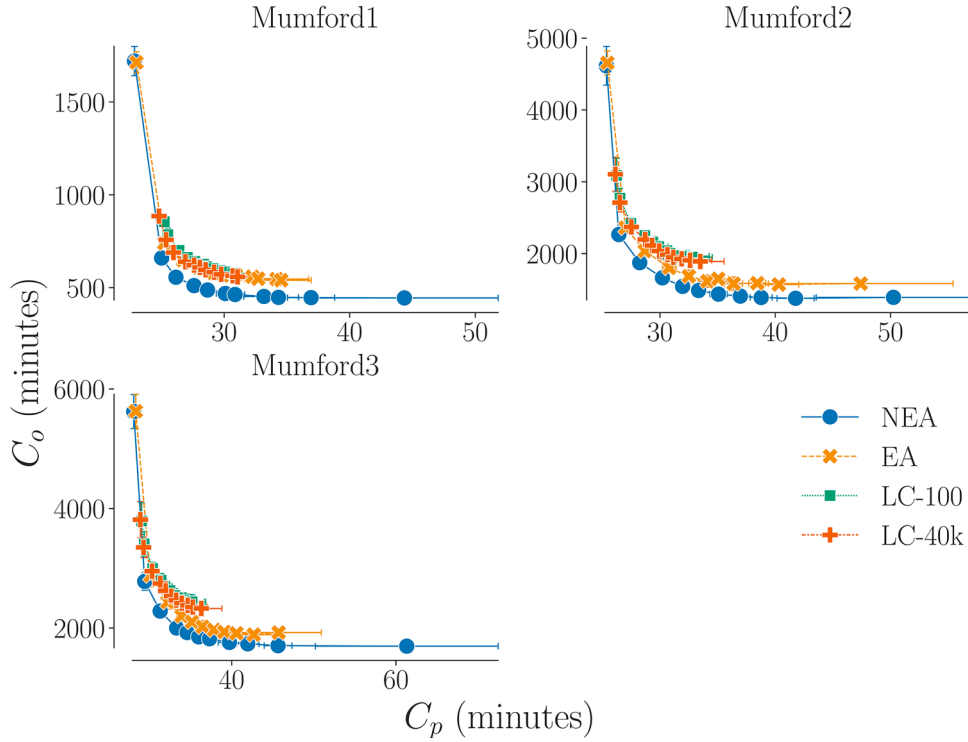


Figure 3. Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis), across values of α over the range $[0, 1]$ in increments of 0.1, from $\alpha = 0.0$ at lower-right to 1.0 at upper-left, for transit networks from LC-100, LC-40k, EA, and NEA. Each point shows 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.

Comparing LC-100 and LC-40k in Figure 3, we see that across all three cities and all values of α , LC-40k improves on LC-100 only slightly by comparison with EA or NEA. From this we conclude that the number of networks considered is not the main cause of EA’s and NEA’s advantage over LC-100: the evolutionary algorithm itself makes most of the difference.

5.2.2. Effect of type-2 heuristic

The type-2 heuristic is the same between EA and NEA and is not a learned function. To understand how important this heuristic is to NEA’s performance, we ran ‘all-1 NEA’, a variant of NEA in which the type-2 heuristic is dropped and only the neural heuristic is used. The results are shown in Figure 4, along with the curve for NEA (reproduced from 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 under-performs NEA, achieving higher C_p (though it also achieves lower C_o , due to the inherent trade-off between C_p and C_o). It appears that the type-2 heuristic’s ability to extend existing routes is important at $\alpha = 1.0$, where making routes longer cannot make the network worse, but otherwise at $\alpha < 1.0$ the neural heuristic does just as well on its own, without the type-2 heuristic.

5.2.3. Effect of reinforcement learning

The type-1 heuristic used in EA and the neural heuristic used in NEA differ in the kind of routes they create. The type-1 heuristic can only add routes that are shortest paths in the link graph, while the neural heuristic may compose multiple shortest paths into a new route. We sought to determine how much of NEA’s advantage over EA is due to this structural difference, versus being due to the specific policy π_θ that we trained using RL.

To answer this question, we ran a variant of NEA in which π_θ is replaced by a purely random policy π_{random} :

$$\pi_{\text{random}}(a | s_t) = \frac{1}{|\mathcal{A}_t|} \quad \forall a \in \mathcal{A}_t \quad (22)$$

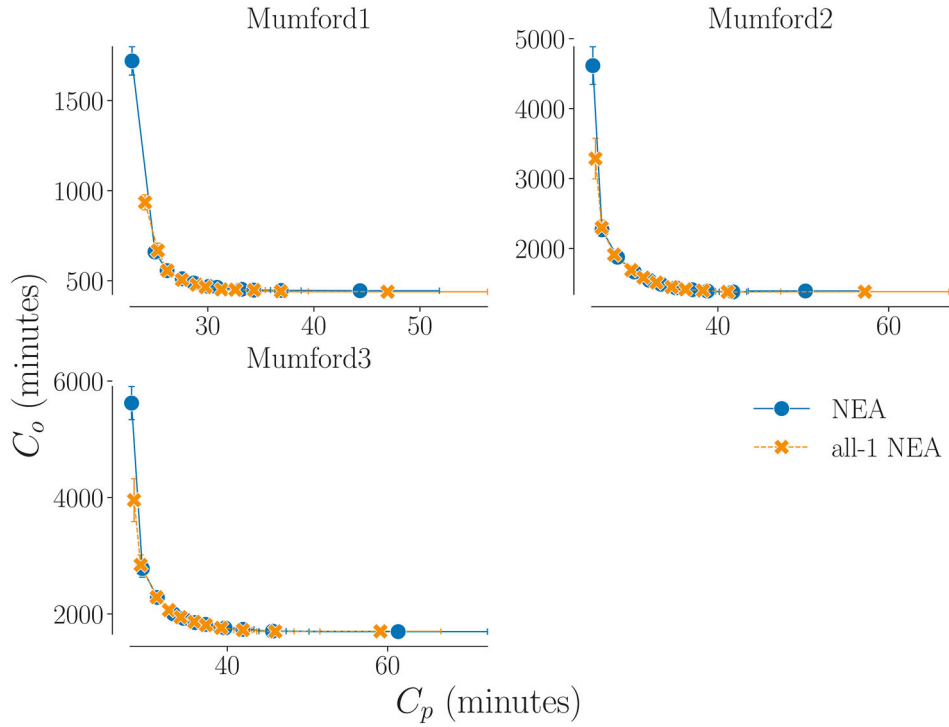


Figure 4. Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis) achieved by all-1 NEA, plotted along with NEA (repeated from Figure 3) for comparison.

We call this variant of NEA the randomly-combining evolutionary algorithm (RC-EA). In order to fully isolate RC-EA from the effects of RL, we also used π_{random} in the LC-100 algorithm to generate the starting network \mathcal{R}_0 for RC-EA.

The results are shown in Figure 5, again with the same NEA curve from Figure 3 for comparison. For values of $\alpha < 1.0$, the transit networks from RC-EA are dominated by some transit network from NEA. But at the extreme of $\alpha = 1.0$, RC-EA’s network performs better than any of NEA’s networks in terms of average trip time C_p , decreasing it by about twenty seconds versus NEA’s lowest- C_p network on each city – an improvement of about 1%.

RC-EA’s poor performance at $\alpha < 1$ makes sense: the 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 a clear disadvantage at $\alpha < 1.0$. The advantage of NEA comes mainly from what the policy π_θ learns during training. It is surprising, though, that RC-EA outperforms NEA at $\alpha = 1.0$. We would expect that the neural net policy used in NEA should be able to equal the performance of RC-EA by learning to choose actions uniformly at random when $\alpha = 1.0$, 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, while noting that composing shortest paths randomly is a good heuristic for minimising C_p at the expense of C_o , if that is one’s aim.

5.3. Comparisons with prior work

We next sought to compare NEA directly against other NDP algorithms from the literature. It is common practice in this literature to report results on the Mandl and Mumford benchmarks only for the operator perspective ($\alpha = 0.0$) and the passenger perspective ($\alpha = 1.0$), so in this section we compare NEA against other algorithms only on these two settings of α .

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. We thus performed a further set of experiments where we ran NEA with $IT = 4,000$ instead of $IT = 400$. In addition, because we observed good results from RC-EA at $\alpha = 1.0$, we ran RC-EA with $IT = 4,000$ at $\alpha = 1.0$, and we include these results in

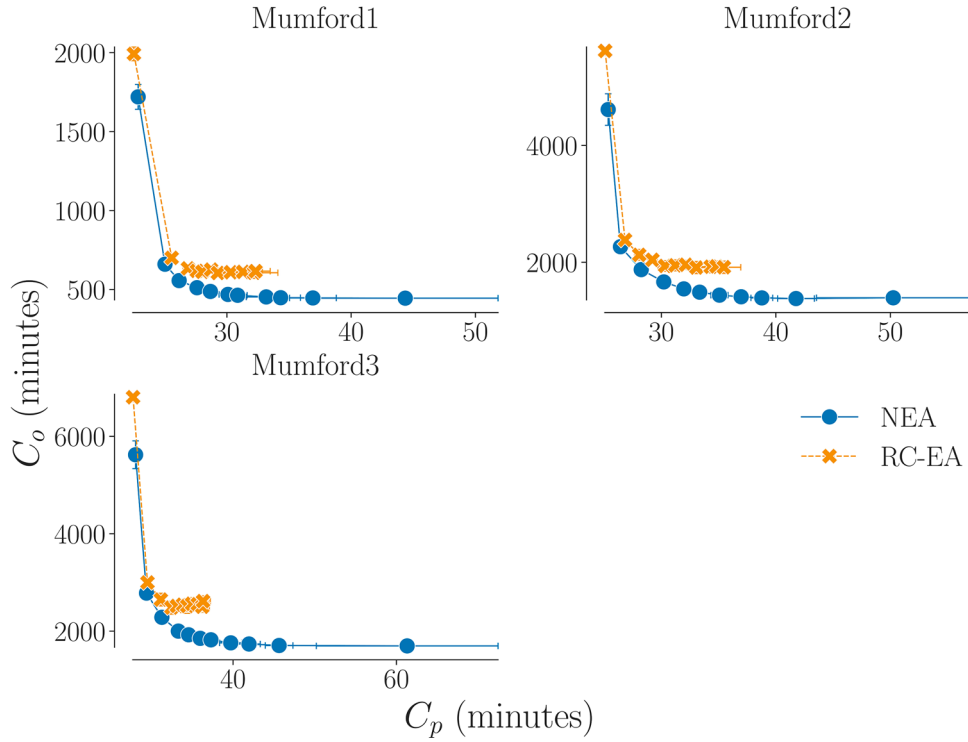


Figure 5. Trade-offs between average trip time C_p (on the x-axis) and total route time C_o (on the y-axis) achieved by RC-EA, plotted along with NEA (repeated from Figure 3) for comparison.

the comparison below. 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 h for each 4,000-iteration run on Mumford3, the largest environment, while RC-EA takes about 6 h.

By comparison, on Mumford3, Mumford (2013a) report no running times; Zervas et al. (2024) run their algorithm for 3 h; Kılıç and Gök (2014) do not report total running time, but take 8 h just to construct the initial network; Ahmed, Mumford, and Kheiri (2019) run their algorithm for 10 h; and Hüsselmann, van Vuuren, and Andersen (2024) and John, Mumford, and Lewis (2014) run theirs for more than two days. So NEA with $IT = 4,000$ falls near the middle of the distribution of computational costs of methods we compare it with, making this a fair comparison.

Tables 3 and 4 present the results of these $IT = 4,000$ runs 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 Subsections 5.1 and 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 d_0 , d_1 , d_2 , and d_{un} , which indicate how many transfers between routes passengers had to make when using the transit network: d_i with $i \in \{0, 1, 2\}$ is the percentage of all passenger trips that required i transfers, while d_{un} is the percentage of trips that require more than two transfers:

$$d_{un} = 100 - (d_0 + d_1 + d_2) \quad (23)$$

5.3.1. Note on omitted work

In these comparisons, we exclude two recent papers that report results on the Mumford benchmark: these are Islam et al. (2019), and Wens and Vansteenwegen (2024). Both works report passenger-perspective results that are computed in non-standard ways: Islam et al. (2019) ignore passenger trips that take more than 2 transfers when computing C_p , while Wens and Vansteenwegen (2024) ignore constraint 1 from the constraints listed in Section 3, allowing the network to fail to connect some pairs of nodes. This means that their reported C_p values are not directly comparable with our own, nor with those reported in other work on the NDP. We exclude them so that the differences between compared results are attributable only to the algorithms that produced them.

Table 3. Passenger-perspective results.

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, Mumford, and Lewis (2014)	10.25	212	–	–	–	–
	Kiliç and Gök (2014)	10.29	216	95.5	4.5	0	0
	Ahmed, Mumford, and Kheiri (2019)	10.18	212	97.17	2.82	0.00	0
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	10.27	179	95.94	3.93	0.13	0
	Hüsselmann, van Vuuren, and Andersen (2024) NSGA-II	10.19	197	97.36	2.64	0	0
	Zervas et al. (2024)	10.19	–	97.94	2.06	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, Mumford, and Lewis (2014)	15.4	745	–	–	–	–
	Kiliç and Gök (2014)	14.99	707	69.73	30.03	0.24	0
	Ahmed, Mumford, and Kheiri (2019)	14.09	722	88.74	11.25	0	0
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	15.48	431	65.5	34.5	0	0
	Hüsselmann, van Vuuren, and Andersen (2024) NSGA-II	14.34	635	86.94	13.06	0	0
	Zervas et al. (2024)	14.76	–	91.0	9.0	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, Mumford, and Lewis (2014)	23.91	1861	–	–	–	–
	Kiliç and Gök (2014)	23.25	1956	45.1	49.08	5.76	0.06
	Ahmed, Mumford, and Kheiri (2019)	21.69	1956	65.75	34.18	0.07	0
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	22.31	1359	57.14	42.63	0.23	0
	Hüsselmann, van Vuuren, and Andersen (2024) NSGA-II	21.94	1851	62.11	37.84	0.05	0
	Zervas et al. (2024)	22.73	–	55.26	43.58	1.16	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, Mumford, and Lewis (2014)	27.02	5461	–	–	–	–
	Kiliç and Gök (2014)	26.82	5027	33.88	57.18	8.77	0.17
	Ahmed, Mumford, and Kheiri (2019)	25.19	5257	56.68	43.26	0.05	0
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	25.65	3583	48.07	51.29	0.64	0
	Hüsselmann, van Vuuren, and Andersen (2024) NSGA-II	25.31	4171	52.56	47.33	0.11	0
	Zervas et al. (2024)	25.71	–	57.84	42.0	0.16	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, Mumford, and Lewis (2014)	29.5	6320	–	–	–	–
	Kiliç and Gök (2014)	30.41	5834	27.56	53.25	17.51	1.68
	Ahmed, Mumford, and Kheiri (2019)	28.05	6119	50.41	48.81	0.77	0
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	28.22	4060	45.07	54.37	0.56	0
	Hüsselmann, van Vuuren, and Andersen (2024) NSGA-II	28.03	5018	48.71	51.1	0.19	0
	Zervas et al. (2024)	28.1	–	61.59	38.17	0.24	0
	NEA	27.96	6127	49.36	50.04	0.60	0
	RC-EA	27.60	6896	50.90	48.99	0.11	0

Note: 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. Bold values in C_p , d_0 , and d_{un} columns are the best for that city.

5.3.2. Results

Table 3 shows the passenger perspective results. On Mandl, Mumford0, and Mumford1, Ahmed, Mumford, and Kheiri (2019)'s method 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, van Vuuren, and Andersen (2024)'s NSGA-II by 0.3%. Meanwhile, RC-EA performs best of all on both Mumford2 and Mumford3, improving on Mumford3's previous best C_p by 1.5%, and setting a new state-of-the-art on these two challenging benchmarks.

Table 4 shows the operator-perspective results. Kiliç and Gök (2014) and Zervas et al. (2024) do not report results for the operator perspective, so are not included here. Similarly to the passenger perspective, NEA here underperforms on the smallest cities (Mandl and Mumford0), but outperforms all methods other than Ahmed, Mumford, and Kheiri (2019) on Mumford1 and Mumford2, including the more recent work of Hüsselmann, van Vuuren, and Andersen (2024). And on the most challenging city, Mumford3, NEA again outperforms all other methods, improving on the previous state-of-the-art C_o by 4.8%.

Table 4. Operator perspective results.

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, Mumford, and Lewis (2014)	63	13.48	—	—	—	—
	Ahmed, Mumford, and Kheiri (2019)	63	14.28	62.23	27.16	9.57	1.03
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	63	13.55	70.99	24.44	4.00	0.58
	Hüsselmann, van Vuuren, and Andersen (2024) 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, Mumford, and Lewis (2014)	95	32.78	—	—	—	—
	Ahmed, Mumford, and Kheiri (2019)	94	26.32	14.61	31.59	36.41	17.37
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	98	27.61	22.39	31.27	18.82	27.51
	Hüsselmann, van Vuuren, and Andersen (2024) 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, Mumford, and Lewis (2014)	462	39.98	—	—	—	—
	Ahmed, Mumford, and Kheiri (2019)	408	39.45	18.02	29.88	31.9	20.19
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	511	26.48	25.17	59.33	14.54	0.96
	Hüsselmann, van Vuuren, and Andersen (2024) 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, Mumford, and Lewis (2014)	1875	32.33	—	—	—	—
	Ahmed, Mumford, and Kheiri (2019)	1330	46.86	13.63	23.58	23.94	38.82
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	1979	29.91	22.77	58.65	18.01	0.57
	Hüsselmann, van Vuuren, and Andersen (2024) 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, Mumford, and Lewis (2014)	2301	36.12	—	—	—	—
	Ahmed, Mumford, and Kheiri (2019)	1746	46.05	16.28	24.87	26.34	32.44
	Hüsselmann, van Vuuren, and Andersen (2024) DBMOSA	2682	32.33	23.55	58.05	17.18	1.23
	Hüsselmann, van Vuuren, and Andersen (2024) 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

Note: 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. Bold values in C_o , d_0 and d_{un} columns are the best for that city.

Despite NEA and RC-EA's relatively simple metaheuristic, these two algorithms set a new state of the art on the most challenging of NDP benchmarks shows that on large city graphs, our neural heuristic is superior to earlier non-neural heuristics. And in the passenger-perspective case, our non-neural heuristic of randomly concatenating shortest paths performs even better. The pattern in these results, where the relative performance of our heuristics increases with the size of the city n , also suggests that their advantage may continue to grow for cities larger than those found in these benchmarks.

When optimising just one metric as in the above experiments, our method achieves low values of that metric by driving the other metric very high, as can be seen in the C_o column of Table 3 and the C_p column of Table 4. This is because of the inherent trade-off between C_p and C_o : shorter, more direct routes reduce operating cost but connect fewer pairs of nodes directly, requiring passengers to make more transfers and increasing the cost to passengers.

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, van Vuuren, and Andersen 2024). 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 minimising operating cost. Meanwhile, at $\alpha = 1.0$, the numbers of transfers achieved by NEA and RC-EA are comparable to other recent methods, except for Zervas et al. (2024). Unlike our method and the others compared here, their algorithm directly incorporates the number of transfers in the cost function that it minimises, resulting in fewer transfers but less impressive C_p .

6. Case study: laval

To assess our method in a realistic case, we applied it to a simulation of the city of Laval in Quebec, Canada. Laval is a suburb of the major Canadian city of Montreal (Figure 6). As of the 2021 census, it had a total population of

Table 5. Statistics and parameters of the Laval scenario.

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

about 430,000 (Statistics Canada 2023). Public transit in Laval is provided primarily by the bus network of the Société de Transport de Laval; additionally, the Orange Line of the Montreal underground metro system has three stops in Laval.

We constructed a city graph to represent Laval based on the 2021 census dissemination areas (CDAs) obtained 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 Origin-Destination (OD) dataset (Agence Métropolitaine de transport 2013) provided by Quebec’s Agence Métropolitaine de Transport, and publicly-available GTFS data from 2013 (Société de Transport de Laval n.d) that describes Laval’s existing transit system.

Each node in the graph represents the centroid of a CDA, with edges connecting it to the nodes of neighbouring CDAs, weighted by the average travel time between them over the road network. The demand matrix D was derived from the OD dataset, which gave an estimated number of trips between every pair of locations based on a large survey of Laval residents. Trips that went outside of Laval to Montreal were redirected to one of several points in the graph where a metro route or bus route in STL’s existing transit system crosses into Montreal. This yielded a graph with 632 nodes. Table 5 contains the statistics of, and parameters used for, the Laval scenario.

For the full details of how the graph was constructed from this data, we refer the reader to chapter 5 of Holliday (2024).

6.1. Existing transit

Laval’s existing transit network has 43 daytime bus routes. To compare networks from our algorithm to this network, we mapped each stop on these routes to the nearest node in the node set \mathcal{N} based on which CDA contains the stop; duplicate stops on the resulting routes were merged. We refer to this translation of the real transit network as the STL network.

The Montreal metro system’s Orange Line has three stops in Laval. As with the bus routes, we mapped these stops to their containing CDAs to form an extra metro route m , which is treated as being present in all simulations. We also included this route in the MDP by setting $\mathcal{R}_0 = \{m\}$, so that our neural policies would take it as input and account for it.

The numbers of stops on the routes in the STL network range from 2 to 52. To ensure a fair comparison, we set the constraint parameters to $S = 44$ (43 plus 1 for the metro line), $MIN = 2$ and $MAX = 52$ when running our algorithm. Unlike the routes in our algorithm’s networks, some of STL’s routes are asymmetric or unidirectional. We allowed this in our simulation of STL, while still requiring symmetry and bidirectionality for the transit networks from our algorithms. For this reason, when reporting total route time C_o in this section, it is calculated by summing the travel times of bidirectional routes in both directions, instead of just one direction as we do elsewhere.

6.2. Enforcing constraint satisfaction

In the experiments on the Mandl and Mumford benchmark cities described in Section 5, the EA and NEA algorithms reliably produced networks that satisfied all of the constraints outlined in Section 3. However, Laval’s city graph, with 632 nodes, is considerably larger than the largest Mumford city, which has only 127 nodes. In our initial experiments on Laval, we found that both EA and NEA consistently failed to satisfy constraint 1, producing networks that left some pairs of nodes disconnected.

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

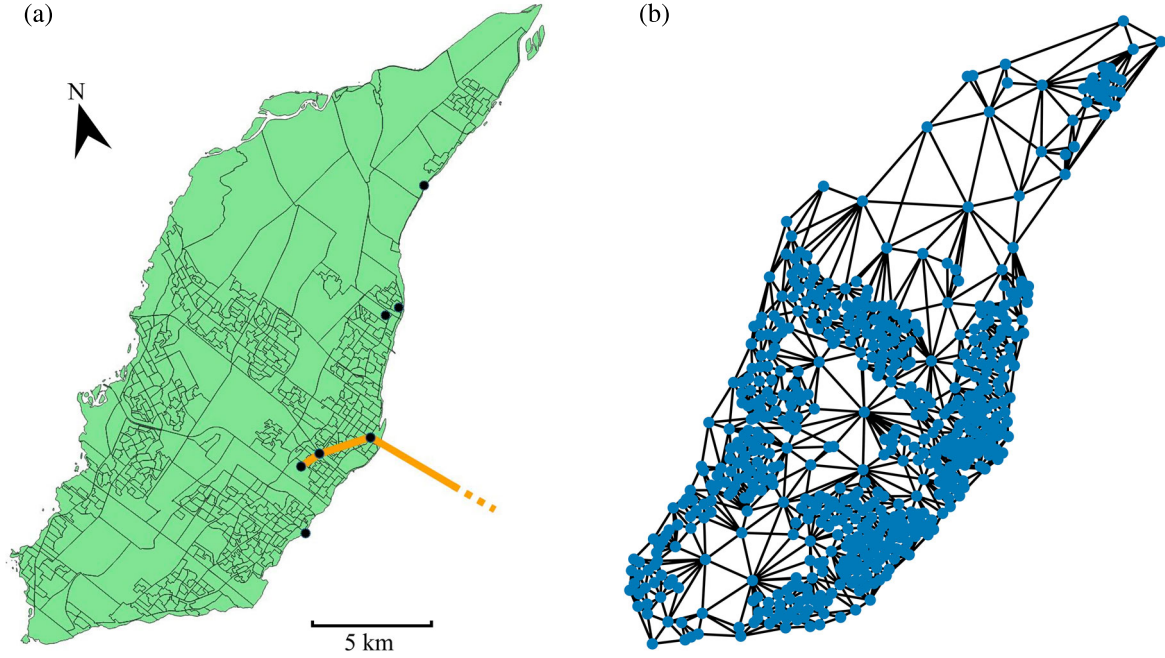


Figure 6. (a) shows a map of the census dissemination areas of the city of Laval, with ‘crossover points’ used to remap inter-city demands shown as black circles, and the Montreal Metro Orange Line shown in orange. (b) shows the link graph constructed from these dissemination areas. (a) Census dissemination areas (b) Link graph.

- When the timestep t is even and $\mathcal{A}_t = \{\text{continue}, \text{halt}\}$, the ‘halt’ action is removed from \mathcal{A}_t . This means that 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 $c_1(\mathcal{G}, \mathcal{R}'_t)$ if added to r_t , then all paths that would *not* reduce $c_1(\mathcal{G}, \mathcal{R}'_t)$ are removed from \mathcal{A}_t . This means that if it is *possible* to meet some unmet demand by extending r_t , then it *must* be done.

With these changes to the MDP, we found that both LC-100 and NEA reliably produced networks that satisfied all constraints, even using policies π_θ trained without these changes. The results reported in this section were obtained using this variant of the MDP.

6.3. Experiments

We evaluated both the NEA and RC-EA algorithms on the Laval city graph. We ran three sets of NEA experiments representing different perspectives: the operator perspective ($\alpha = 0.0$), the passenger perspective ($\alpha = 1.0$), and a balanced perspective ($\alpha = 0.5$). We only ran RC-EA at $\alpha = 1.0$ because as we observed previously in Subsection 5.2.3, RC-EA performs poorly at $\alpha < 1.0$. As in Section 5, we perform ten runs of each experiment with ten different random seeds and the same ten different learned policies π_θ as in our other experiments, and we report statistics over these ten runs.

We kept the same parameters of NEA and RC-EA that we used in Subsection 5.1: $IT = 400$, $B = 10$, $E = 10$, and a 300s transfer penalty. Although the computer memory requirements of running our algorithms on this very large graph were greater than for the Mumford benchmark datasets, the desktop machine used for our Mumford experiments proved adequate for the Laval experiments as well, and no changes to our model or algorithm were required to reduce memory usage.

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 NEA improves on these initial networks. Figure 7 highlights the trade-offs each method achieves between average trip time C_p and total route time C_o .

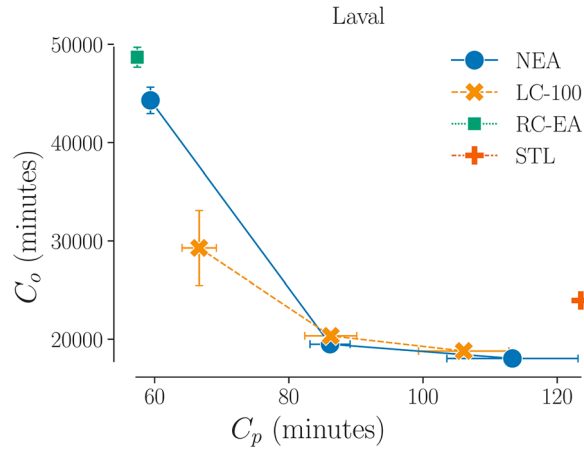


Figure 7. Average trip time C_p versus total route time C_o achieved by LC-100 and NEA at $\alpha = 0.0$ (bottom-right), 0.5 (middle), and 1.0 (top-left), as well as for RC-EA at $\alpha = 1.0$ and the STL network (independent of α). Except for STL, each point is a mean value over 10 random seeds, and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

Table 6. Performance of LC-100 and NEA's networks at three α values, RC-EA's networks at $\alpha = 1.0$, and the STL network.

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 \pm 6.74	18797 \pm 190	15.45 \pm 0.54	22.98 \pm 1.34	23.71 \pm 1.08	37.86 \pm 1.87
	NEA	113.33 \pm 9.78	18037 \pm 302	15.44 \pm 0.47	22.50 \pm 1.72	23.23 \pm 1.27	38.83 \pm 2.67
0.5	LC-100	86.25 \pm 3.88	20341 \pm 549	15.04 \pm 0.46	26.46 \pm 1.10	26.76 \pm 1.18	31.74 \pm 2.22
	NEA	86.14 \pm 2.99	19494 \pm 389	14.89 \pm 0.62	26.18 \pm 1.53	26.59 \pm 1.69	32.34 \pm 2.90
1.0	LC-100	66.64 \pm 2.56	29286 \pm 3815	17.16 \pm 1.27	34.50 \pm 3.49	29.03 \pm 1.00	19.30 \pm 4.81
	NEA	59.38 \pm 0.52	44314 \pm 1344	22.08 \pm 0.71	38.79 \pm 2.03	27.57 \pm 0.75	11.56 \pm 2.22
	RC-EA	57.40 \pm 0.17	48700 \pm 1001	23.73 \pm 0.57	44.31 \pm 0.71	25.20 \pm 0.62	6.76 \pm 0.58

Note: Bold values are the best for the corresponding α , except where STL performs best at that α , in which case no values are bold. 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 ' \pm ' where relevant.

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 operating cost C_o at $\alpha = 0.0$, and at $\alpha = 0.5$ it achieves 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 optimised, 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 Subsection 5.1: NEA's improvements over LC-100 tend to be greater in C_o than in C_p . RC-EA, meanwhile, improves on NEA in terms of C_p at $\alpha = 1.0$, 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. At $\alpha = 1.0$ RC-EA achieves the fewest transfers. But at both $\alpha = 0.5$ and $\alpha = 1.0$, NEA also decreases the overall number of transfers relative to the STL network.

The likely aim of Laval's transit agency is to reduce costs while improving service quality, so the 'balanced' case, where $\alpha = 0.5$, is the most relevant in reality. Some simple calculations can give us a rough estimate of savings NEA offers in this case. The approximate cost of operating a Laval city bus is 200 Canadian dollars (CAD) per hour. If each route has the same headway (time between bus arrivals) h , the number of buses N_r required by route r is $\lceil \frac{\tau_r}{h} \rceil$. The total number of buses $N_{\mathcal{R}}$ required for a transit network \mathcal{R} is then:

$$N_{\mathcal{R}} = \sum_{r \in \mathcal{R}} N_r \approx \frac{1}{h} \sum_{r \in \mathcal{R}} \tau_r = \frac{C_o}{h} \quad (24)$$

The total cost of the system is then $200 \text{ CAD} \times N_{\mathcal{R}}$. Assuming a headway of 15 minutes on all routes, that gives a per-hour operating cost of 319,387 CAD for the STL network. By comparison, the networks from NEA with $\alpha = 0.5$ cost 259,920 CAD per hour on average, saving the transit agency 59,500 CAD per hour.

7. Discussion

The choice of heuristics has a major impact on the effectiveness of a metaheuristic algorithm, and our results show that neural heuristics trained by deep reinforcement learning can outperform human-engineered heuristics on the transit network design problem. Neural heuristics can also be useful in complex real-world transit planning scenarios, where they can improve an existing transit system in multiple dimensions.

Our heuristic-training method has several limitations. One is that the construction process on which our neural net policies are trained differs from the evolutionary algorithm – an improvement process – in which it is deployed. We would like to train our neural net policies directly in the context of an improvement process, which may make them better-suited to that process.

Another limitation is that we only train neural net policies to make one kind of change to a transit network: concatenating multiple shortest paths into a new route and adding it to the network. In future, we wish to train a more diverse set of policies, perhaps including route-lengthening and -shortening operators that also could be included as heuristics in a metaheuristic algorithm.

It would also be worthwhile to apply our neural heuristics within other metaheuristics than just the evolutionary algorithm developed here. Multi-objective metaheuristic algorithms could use neural heuristics in tandem with Pareto optimal solution methods such as TOPSIS (Papathanasiou and Ploskas 2018) to find optimal transit networks.

One remaining question is why the random path-combining heuristic used in the RC-EA experiments outperforms the neural net heuristic for the passenger perspective. As noted in Subsection 5.2.3, in principle the neural net should be able to learn to imitate a random policy if that gives the best outcome. 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 neural heuristic's performance to match or exceed that of the random policy.

Our aim in this paper was to show whether neural 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.

Note

1. Available at https://github.com/aholliday/transit_learning.

Acknowledgements

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

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by NSERC under Grant 223292.

ORCID

Andrew Holliday  <http://orcid.org/0000-0002-1507-6505>

References

- Agence Métropolitaine de Transport. 2013. "Enquête Origine-Destination 2013." Montreal, QC.
- Ahmed, Leena, Christine Mumford, and Ahmed Kheiri. 2019. "Solving Urban Transit Route Design Problem Using Selection Hyper-heuristics." *European Journal of Operational Research* 274 (2): 545–559. <https://doi.org/10.1016/j.ejor.2018.10.022>.
- Ai, Guanqun, Xingquan Zuo, Gang Chen, and Binglin Wu. 2022. "Deep Reinforcement Learning Based Dynamic Optimization of Bus Timetable." *Applied Soft Computing* 131:109752. <https://doi.org/10.1016/j.asoc.2022.109752>.
- Aksoy, Ilyas Cihan, and Mehmet Metin Mutlu. 2024. "Comparing the Performance of Metaheuristics on the Transit Network Frequency Setting Problem." *Journal of Intelligent Transportation Systems* 28: 1–20. <https://doi.org/10.1080/15472450.2024.2392722>.
- Applegate, David, Robert E. Bixby, Vašek Chvátal, and William J. Cook. 2001. "Concorde TSP Solver." <https://www.math.uwaterloo.ca/tsp/concorde/index.html>.
- Battaglia, Peter W., Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, et al. 2018. "Relational Inductive Biases, Deep Learning, and Graph Networks." Preprint [arXiv:1806.01261](https://arxiv.org/abs/1806.01261).
- Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost. 2021. "Machine Learning for Combinatorial Optimization: A Methodological Tour D'hORIZON." *European Journal of Operational Research* 290 (2): 405–421. <https://doi.org/10.1016/j.ejor.2020.07.063>.
- Brody, Shaked, Uri Alon, and Eran Yahav. 2021. "How Attentive Are Graph Attention Networks?" <https://arxiv.org/abs/2105.14491>.
- Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. "Spectral Networks and Locally Connected Networks on Graphs." Preprint [arXiv:1312.6203](https://arxiv.org/abs/1312.6203).
- Chen, Xinyun, and Yuandong Tian. 2019. "Learning to Perform Local Rewriting for Combinatorial Optimization." *Advances in Neural Information Processing Systems* 32.
- Chien, Steven I-Jy, Yuqing Ding, and Chienhung Wei. 2002. "Dynamic Bus Arrival Time Prediction with Artificial Neural Networks." *Journal of Transportation Engineering* 128 (5): 429–438. [https://doi.org/10.1061/\(ASCE\)0733-947X\(2002\)128:5\(429\)](https://doi.org/10.1061/(ASCE)0733-947X(2002)128:5(429)).
- Choo, Jinho, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. 2022. "Simulation-Guided Beam Search for Neural Combinatorial Optimization." *Advances in Neural Information Processing Systems* 35:8760–8772.
- Çodur, Muhammed Yasin, and Ahmet Tortum. 2009. "An Artificial Intelligent Approach to Traffic Accident Estimation: Model Development and Application." *Transport* 24 (2): 135–142. <https://doi.org/10.3846/1648-4142.2009.24.135-142>.
- da Costa, Paulo, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. 2020. "Learning 2-opt Heuristics for the Traveling Salesman Problem via Deep Reinforcement Learning." In *Asian Conference on Machine Learning*, 465–480. Bangkok, Thailand: PMLR.
- Dai, Hanjun, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. "Learning Combinatorial Optimization Algorithms over Graphs." Preprint [arXiv:1704.01665](https://arxiv.org/abs/1704.01665).
- Darwish, Ahmed, Momen Khalil, and Karim Badawi. 2020. "Optimising Public Bus Transit Networks Using Deep Reinforcement Learning." In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 1–7. Rhodes, Greece: IEEE.
- Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. 2016. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering." *CoRR*. <http://arxiv.org/abs/1606.09375>.
- Durán-Micco, Javier, and Pieter Vansteenwegen. 2022. "A Survey on the Transit Network Design and Frequency Setting Problem." *Public Transport* 14 (1): 155–190. <https://doi.org/10.1007/s12469-021-00284-y>.
- Duvenaud, David K., Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. "Convolutional Networks on Graphs for Learning Molecular Fingerprints." *Advances in Neural Information Processing Systems* 28.
- Fortune, Steven. 1995. "Voronoi Diagrams and Delaunay Triangulations." In *Computing in Euclidean Geometry*, 225–265. Boca Raton, Florida: CRC Press.
- Fu, Zhang-Hua, Kai-Bin Qiu, and Hongyuan Zha. 2021. "Generalize A Small pre-trained Model to Arbitrarily Large TSP Instances." In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, 7474–7482.
- Gilmer, Justin, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. "Neural Message Passing for Quantum Chemistry." In *Proceedings of the 34th International Conference on Machine Learning*, edited by Doina Precup and Yee Whye Teh, Vol. 70 of *Proceedings of Machine Learning Research*, 1263–1272. Sydney, Australia: PMLR.
- Guan, J. F., Hai Yang, and S. C. Wirasinghe. 2006. "Simultaneous Optimization of Transit Line Configuration and Passenger Line Assignment." *Transportation Research Part B: Methodological* 40:885–902. <https://doi.org/10.1016/j.trb.2005.12.003>.
- Guihaire, Valérie, and Jin-Kao Hao. 2008. "Transit Network Design and Scheduling: A Global Review." *Transportation Research Part A: Policy and Practice* 42 (10): 1251–1273.
- Holliday, Andrew, and Gregory Dudek. 2023. "Augmenting Transit Network Design Algorithms with Deep Learning." In *2023 26th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. Bilbao, Spain: IEEE.
- Holliday, Andrew, and Gregory Dudek. 2024. "A Neural-Evolutionary Algorithm for Autonomous Transit Network Design." In *Presented at 2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE.

- Holliday, Andrew. 2024. "Applications of Deep Reinforcement Learning to Urban Transit Network Design." PhD diss., McGill University.
- Hottung, André, and Kevin Tierney. 2019. "Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem." Preprint [arXiv:1911.09539](https://arxiv.org/abs/1911.09539).
- Hüsselmann, Günther, Jan Harm van Vuuren, and Simen Johann Andersen. 2024. "An Improved Solution Methodology for the Urban Transit Routing Problem." *Computers & Operations Research* 163:106481. <https://doi.org/10.1016/j.cor.2023.106481>.
- Islam, Kazi Ashik, Ibraheem Muhammad Moosa, Jaiaid Mobin, Muhammad Ali Nayeem, and M. Sohel Rahman. 2019. "A Heuristic Aided Stochastic Beam Search Algorithm for Solving the Transit Network Design Problem." *Swarm and Evolutionary Computation* 46:154–170. <https://doi.org/10.1016/j.swevo.2019.02.007>.
- Jeong, Ranhee, and R. Rilett. 2004. "Bus Arrival Time Prediction Using Artificial Neural Network Model." In *Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No. 04TH8749)*, 988–993. Washington, DC: IEEE.
- Jiang, Zhibin, Wei Fan, Wei Liu, Bingqin Zhu, and Jinjing Gu. 2018. "Reinforcement Learning Approach for Coordinated Passenger Inflow Control of Urban Rail Transit in Peak Hours." *Transportation Research Part C: Emerging Technologies* 88:1–16. <https://doi.org/10.1016/j.trc.2018.01.008>.
- John, Matthew P., Christine L. Mumford, and Rhyd Lewis. 2014. "An Improved Multi-objective Algorithm for the Urban Transit Routing Problem." In *Evolutionary Computation in Combinatorial Optimisation*, edited by Christian Blum and Gabriela Ochoa, 49–60. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Kar, Armita, Andre L. Carrel, Harvey J. Miller, and Huyen T. K. Le. 2022. "Public Transit Cuts during COVID-19 Compound Social Vulnerability in 22 US Cities." *Transportation Research Part D: Transport and Environment* 110:103435. <https://doi.org/10.1016/j.trd.2022.103435>.
- Kepaptsoglou, Konstantinos, and Matthew Karlaftis. 2009. "Transit Route Network Design Problem: Review." *Journal of Transportation Engineering* 135 (8): 491–505. [https://doi.org/10.1061/\(ASCE\)0733-947X\(2009\)135:8\(491\)](https://doi.org/10.1061/(ASCE)0733-947X(2009)135:8(491)).
- Kim, Minsu, Jinkyoo Park, and Joungho Kim. 2021. "Learning Collaborative Policies to Solve NP-hard Routing Problems." *Advances in Neural Information Processing Systems* 34:10418–10430.
- Kingma, Diederik P., and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." In *ICLR*. San Diego, California.
- Kipf, Thomas N., and Max Welling. 2016. "Semi-supervised Classification with Graph Convolutional Networks." Preprint [arXiv:1609.02907](https://arxiv.org/abs/1609.02907).
- Kool, W., H. V. Hoof, and M. Welling. 2019. "Attention, Learn to Solve Routing Problems!" In *ICLR*. New Orleans, Louisiana.
- Kılıç, Fatih, and Mustafa Gök. 2014. "A Demand Based Route Generation Algorithm for Public Transit Network Design." *Computers & Operations Research* 51:21–29. <https://doi.org/10.1016/j.cor.2014.05.001>.
- Li, Can, Lei Bai, Wei Liu, Lina Yao, and S. Travis Waller. 2020. "Graph Neural Network for Robust Public Transit Demand Prediction." *IEEE Transactions on Intelligent Transportation Systems* 23:4086–4098. <https://doi.org/10.1109/TITS.2020.3041234>.
- Li, Can, Lei Bai, Lina Yao, S. Travis Waller, and Wei Liu. 2023. "A Bibliometric Analysis and Review on Reinforcement Learning for Transportation Applications." *Transportmetrica B: Transport Dynamics* 11 (1):2179461.
- Lin, Haifeng, and Chengpei Tang. 2022. "Analysis and Optimization of Urban Public Transport Lines Based on Multiobjective Adaptive Particle Swarm Optimization." *IEEE Transactions on Intelligent Transportation Systems* 23 (9): 16786–16798. <https://doi.org/10.1109/TITS.2021.3086808>.
- Liu, Luyu, Harvey J. Miller, and Jonathan Scheff. 2020. "The Impacts of COVID-19 Pandemic on Public Transit Demand in the United States." *PloS One* 15 (11): e0242476. <https://doi.org/10.1371/journal.pone.0242476>.
- Ma, Yining, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. "Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer." *Advances in Neural Information Processing Systems* 34:11096–11107.
- Mandl, Christoph E. 1980. "Evaluation and Optimization of Urban Public Transportation Networks." *European Journal of Operational Research* 5 (6): 396–404. [https://doi.org/10.1016/0377-2217\(80\)90126-5](https://doi.org/10.1016/0377-2217(80)90126-5).
- Mirhoseini, Azalia, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, et al. 2021. "A Graph Placement Methodology for Fast Chip Design." *Nature* 594 (7862): 207–212. <https://doi.org/10.1038/s41586-021-03544-w>.
- Mumford, Christine L. 2013a. "New Heuristic and Evolutionary Operators for the Multi-objective Urban Transit Routing Problem." In *2013 IEEE Congress on Evolutionary Computation*, 939–946. Cancun, Mexico: IEEE.
- Mumford, Christine L. 2013b. "Supplementary Material for: New Heuristic and Evolutionary Operators for the Multi-objective Urban Transit Routing Problem, CEC 2013." Accessed March 24, 2023. <https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp.zip>.
- Mundhenk, T. Nathan, Mikel Landajuela, Ruben Glatt, Claudio P. Santiago, Daniel M. Faissol, and Brenden K. Petersen. 2021. "Symbolic Regression via Neural-Guided Genetic Programming Population Seeding." Preprint [arXiv:2111.00053](https://arxiv.org/abs/2111.00053).
- Nikolić, Miloš, and Dušan Teodorović. 2013. "Transit Network Design by Bee Colony Optimization." *Expert Systems with Applications* 40 (15): 5945–5955. <https://doi.org/10.1016/j.eswa.2013.05.002>.
- Papathanasiou, Jason, and Nikolaos Ploskas. 2018. *Multiple Criteria Decision Aid: Methods, Examples and Python Implementations*. Cham, Switzerland: Springer International Publishing.
- Rodrigue, Jean-Paul. 1997. "Parallel Modelling and Neural Networks: An Overview for Transportation/land Use Systems." *Transportation Research Part C: Emerging Technologies* 5 (5): 259–271. [https://doi.org/10.1016/S0968-090X\(97\)00014-4](https://doi.org/10.1016/S0968-090X(97)00014-4).

- Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. "Proximal Policy Optimization Algorithms." Preprint [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- Société de Transport de Laval. n.d.. "GTFS Data for the Société de Transport de Laval Bus Network." Retrieved 2020. <https://transitfeeds.com/p/societe-de-transport-de-laval/38/1383528159>.
- Sörensen, Kenneth, Marc Sevaux, and Fred Glover. 2018. "A History of Metaheuristics." In *Handbook of Heuristics*, 791–808. Cham, Switzerland: Springer.
- Sörensen, Kenneth. 2015. "Metaheuristics—the Metaphor Exposed." *International Transactions in Operational Research* 22 (1): 3–18. <https://doi.org/10.1111/itor.2015.22.issue-1>.
- Statistics Canada. 2021. "Census Dissemination Area Boundary Files." Accessed July 1, 2023. <https://www150.statcan.gc.ca/n1/en/catalogue/92-169-X>.
- Statistics Canada. 2023. "Census Profile, 2021 Census of Population." Accessed October 25, 2023. <https://www12.statcan.gc.ca/census-recensement/2021/dp-pd/prof/details/page.cfm>.
- Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. London, England: MIT Press.
- Sykora, Quinlan, Mengye Ren, and Raquel Urtasun. 2020. "Multi-agent Routing Value Iteration Network." In *International Conference on Machine Learning*, 9300–9310. PMLR.
- van Nes, Rob. 2003. "Multiuser-Class Urban Transit Network Design." *Transportation Research Record* 1835 (1): 25–33. <https://doi.org/10.3141/1835-04>.
- Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. 2015. "Pointer Networks." Preprint [arXiv:1506.03134](https://arxiv.org/abs/1506.03134).
- Wang, Tao, Zhipeng Zhu, Jing Zhang, Junfang Tian, and Wenyi Zhang. 2024. "A Large-Scale Traffic Signal Control Algorithm Based on Multi-layer Graph Deep Reinforcement Learning." *Transportation Research Part C: Emerging Technologies* 162:104582. <https://doi.org/10.1016/j.trc.2024.104582>.
- Wei, Yu, Minjia Mao, Xi Zhao, Jianhua Zou, and Ping An. 2020. "City Metro Network Expansion with Reinforcement Learning." In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2646–2656.
- Wens, Maarten, and Pieter Vansteenwegen. 2024. "A Decentralised Genetic Algorithm with sub-objectives for the Transit Network Design Problem." *Computers & Operations Research* 183:107191. <https://doi.org/10.1016/j.cor.2025.107191>.
- Williams, Ronald J. 1992. "Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning." *Machine Learning* 8 (3): 229–256. <https://doi.org/10.1023/A:1022672621406>.
- Wu, Fei, Changjiang Zheng, Muqing Du, Wei Ma, and Junze Ma. 2025. "Heterogeneous Multi-view Graph Gated Neural Networks for Real-Time Origin-Destination Matrix Prediction in Metro Systems." *Transportmetrica B: Transport Dynamics* 13 (1):2449483.
- Wu, Yaixin, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2021. "Learning Improvement Heuristics for Solving Routing Problems." *IEEE Transactions on Neural Networks and Learning Systems* 33 (9): 5057–5069. <https://doi.org/10.1109/TNNLS.2021.3068828>.
- Xiong, Yihua, and Jerry B. Schneider. 1992. "Transportation Network Design Using a Cumulative Genetic Algorithm and Neural Network." *Transportation Research Record* 1364.
- Yan, Haoyang, Zhiyong Cui, Xinqiang Chen, and Xiaolei Ma. 2023. "Distributed Multiagent Deep Reinforcement Learning for Multiline Dynamic Bus Timetable Optimization." *IEEE Transactions on Industrial Informatics* 19:469–479. <https://doi.org/10.1109/TII.2022.3158651>.
- Yang, Jie, and Yangsheng Jiang. 2020. "Application of Modified NSGA-II to the Transit Network Design Problem." *Journal of Advanced Transportation* 2020: 1–24.
- Ye, Haoran, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. 2024. "DeepACO: Neural-Enhanced Ant Systems for Combinatorial Optimization." *Advances in Neural Information Processing Systems* 36.
- Ying, Rex, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems." *CoRR* [http://arxiv.org/abs/1806.01973](https://arxiv.org/abs/1806.01973).
- Yoo, Sunhyung, Jinwoo Brian Lee, and Hoon Han. 2023. "A Reinforcement Learning Approach for Bus Network Design and Frequency Setting Optimisation." *Public Transport* 15:503–534. <https://doi.org/10.1007/s12469-022-00319-y>.
- Zervas, Alexandros, Christina Iliopoulou, Ioannis Tassopoulos, and Grigorios Beligiannis. 2024. "Solving Large-Scale Instances of the Urban Transit Routing Problem with a Parallel Artificial Bee Colony-Hill Climbing Optimization Algorithm." *Applied Soft Computing* 167:112335. <https://doi.org/10.1016/j.asoc.2024.112335>.
- Zhang, Xinshao, Zhaocheng He, Yiting Zhu, and Linlin You. 2023. "DRL-based Adaptive Signal Control for Bus Priority Service under Connected Vehicle Environment." *Transportmetrica B: Transport Dynamics* 11 (1): 1455–1477.
- Zou, Liang, Jian-min Xu, and Ling-xiang Zhu. 2006. "Light Rail Intelligent Dispatching System Based on Reinforcement Learning." In *2006 International Conference on Machine Learning and Cybernetics*, 2493–2496. Dalian, China.

Appendices

Appendix 1. Neural net architecture

In this appendix, we give a detailed description of the architecture of the neural net policy that is used in our neural heuristic. 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.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 link graph.

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

- (1) D_{ij} , the demand between the nodes
- (2) $s_{ij} = 1$ if $(i, j, \tau_{ij}) \in \mathcal{E}_s$, 0 otherwise
- (3) τ_{ij} if $(i, j, \tau_{ij}) \in \mathcal{E}_s$, 0 otherwise
- (4) $c_{ij} = 1$ if \mathcal{R} links i to j , 0 otherwise
- (5) $c_{0ij} = 1$ if j can be reached from i over \mathcal{R} with no transfers, 0 otherwise
- (6) $c_{1ij} = 1$ if one transfer is needed to reach j from i over \mathcal{R} , 0 otherwise
- (7) $c_{2ij} = 1$ if two transfers are needed to reach j from i over \mathcal{R} , 0 otherwise
- (8) $q_{ij} = 1$ if $i = j$, 0 otherwise
- (9) $\tau_{\mathcal{R}ij}$ if \mathcal{R} links i to j , 0 otherwise
- (10) τ_{rij} if j can be reached from i over \mathcal{R} with no transfers, where r is the route that provides the shortest direct trip between i and j . If i and j are not linked with no transfers, has value 0
- (11) T_{ij} , the shortest-path driving time between i and j
- (12) α
- (13) $1 - \alpha$

The global state vector \mathbf{s}_t at timestep t is composed of:

- (1) $C_p(\mathcal{G}, \mathcal{R}_t \cup \{r_t\})$, the average passenger trip time for the current route set
- (2) $C_o(\mathcal{G}, \mathcal{R}_t \cup \{r_t\})$, the total route time for the current route set
- (3) $|\mathcal{R}_t|$, the number of routes planned so far
- (4) $S - |\mathcal{R}_t|$, the number of routes left to plan
- (5) F_{unr} , the fraction of node pairs with demand $d_{ij} > 0$ that are not connected by $\mathcal{R}_t \cup \{r_t\}$
- (6) α
- (7) $1 - \alpha$

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

A.2 Backbone network

The backbone is a graph attention net composed of five GATv2 (Brody, Alon, and Yahav 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 an $n \times d_{\text{embed}}$ matrix of node embeddings Y . These node embeddings are used by the two policy heads to compute action probabilities.

A.3 Halting module

NN_{halt} is a simple feed-forward MLP with 2 hidden layers. It takes as input \mathbf{s}_t , τ_{r_t} , and \mathbf{y}_i and \mathbf{y}_j , where i and j are the first and last nodes on r_t . It outputs a scalar h , to which we apply the sigmoid function to get the halting policy:

$$\pi(\text{halt} | s_t) = \sigma(h) = \frac{1}{1 + e^{-h}} \quad (\text{A1})$$

$$\pi(\text{continue} | s_t) = 1 - \pi(\text{halt} | s_t) \quad (\text{A2})$$

A.4 Extension module

The extension module NN_{ext} computes the policy on odd-numbered timesteps, when extensions are selected for routes. It consists of two components. The first is an MLP called $\text{MLP}_{\text{ext}1}$, that computes the node-pair scores o_{aij} that are used to calculate the probability of selecting each extension path a to the current route r_t . It takes as input the time between i and j along the path p being considered, τ_{pij} ; the node embeddings \mathbf{y}_i and \mathbf{y}_j , and the edge feature \mathbf{e}_{ij} , and outputs a score o_{pij} . When considering extending an existing route r_t by a path a , that act would connect each pair of nodes on a , as well as each node already on r_t to each node on a . So the score o_a is just the sum of the node-pair scores for the node pairs that

Table A1. 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

this choice would link together on the route:

$$o_{(r|a)} = \sum_{i \in a, j \in a, i \neq j} o_{aij} + \sum_{i \in r, j \in a} o_{(r|a)ij} \quad (\text{A3})$$

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 so depends on $o_{(r|a)}$. We account for this with a second MLP, called MLP_{ext2} , that takes as input \mathbf{s}_t , τ_a , and the previous score $o_{(r|a)}$, and outputs a final scalar score \hat{o}_a for each candidate path. The extension policy is then the softmax of these values:

$$\pi_{\theta}(a|s) = \frac{e^{\hat{o}_a}}{\sum_{a' \in \mathcal{A}} e^{\hat{o}_{a'}}} \quad (\text{A4})$$

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

A.5 Value network

The PPO training algorithm requires a learnable value function $V(s_t)$ to compute the advantage $A_t = G_t^H - V(s_t)$ during training. We parameterise $V(s_t)$ as an MLP with 2 hidden layers of dimension 36. The inputs to $V(s_t)$ are the average of the node feature vectors $\frac{\sum_i \mathbf{x}_i}{n}$, the total demand $\sum_{i,j} D_{ij}$, the means and standard deviations of the elements of D and T , the cost component weight α , and 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 offered no benefit over the version described here.

Appendix 2. Training hyper-parameters

Training of the neural net policies is performed using the PPO reinforcement learning algorithm in conjunction with the well-known Adam optimiser (Kingma and Ba 2015) to update the neural net's parameters based on the losses given by PPO. Table A1 gives the hyper-parameters used during training. We augment the training dataset by applying a set of random transformations in each iteration of training. 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 α is sampled, it has equal probability of being set to 0.0, set to 1.0, or 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 where only C_p or only C_o matters.

Although it is typical to use an entropy-maximisation term in the loss function when using PPO, we found better results by dropping this term, so we do not use it when training out policies.

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