Neural Bee Colony Optimization: A Case Study in Public Transit Network Design

Andrew Holliday
School of Computer Science
McGill University
Montreal, QC
ahollid@cim.mcgill.ca

Gregory Dudek
School of Computer Science
McGill University
dudek@cim.mcgill.ca

Abstract

In this work we explore the combination of metaheuristics and learned neural network solvers for combinatorial optimization. We do this in the context of the **transit network design problem**, a uniquely challenging combinatorial optimization problem with real-world importance. We train a neural network policy to perform single-shot planning of individual transit routes, and then incorporate it as one of several sub-heuristics in a modified Bee Colony Optimization (BCO) metaheuristic algorithm. Our experimental results demonstrate that this hybrid algorithm outperforms the learned policy alone by up to 20% and the original BCO algorithm by up to 53% on realistic problem instances. We perform a set of ablations to study the impact of each component of the modified algorithm.

1 Introduction

The design of urban transit networks is an important real-world problem, but is computationally very challenging. It has some similarities with other combinatorial optimization (CO) problems such as the Travelling Salesman problem (TSP) and Vehicle Routing Problem (VRP), but due to its many-to-many nature, combined with the fact that demand can be satisfied by transfers between transit lines, the problem is much more complex than those well-studied problems. The most successful approaches to the Transit Network Design Problem (NDP) to-date have been metaheuristic algorithms. Metaheuristics are high-level approximate strategies for problem-solving that are agnostic to the kind of problem. Many are inspired by natural phenomena, such as Simulated Annealing (SA), Genetic Algorithm (GA), and Bee Colony Optimization (BCO).

Metaheuristic algorithms have proven useful and remain the state-of-the-art in several very complex optimization problems [Ahmed et al., 2019]. But little cross-over exists between the literature on this problem and that of machine learning with neural networks. In this work, we use a neural network system to learn low-level heuristics for the NDP, and use these learned heuristics in a metaheuristic algorithm. We show that this synthesis of a machine learning approach and meta-heuristic approach outperforms either of them alone.

We first develop a novel Graph Neural Network (GNN) policy model and train it in an Reinforcement Learning (RL) context to output transit networks that minimize an established cost function. We compare the performance of the trained GNN model to that of Nikolić and Teodorović [2013]'s BCO approach on a standard benchmark of NDP instances [Mumford, 2013a], characterizing them over a range of different cost functions. We then integrate this model into a metaheuristic algorithm called BCO, as one of the heuristics that the algorithm can employ as it performs a stochastic search of the solution space. We compare this approach to the GNN model and the unmodified BCO algorithm, and we find that on realistically-sized problem instances, the combination outperforms the GNN by

Preprint. Under review.

up to 20% and BCO by up to 53%. Lastly, we perform several ablations to understand the importance of different components of the proposed system to its performance.

2 Related Work

2.1 Graph Networks and Reinforcement Learning for Optimization Problems

Graph Neural Networks (GNNs) are neural network models that are designed to operate on graph-structured data [Bruna et al., 2013, Kipf and Welling, 2016, Defferrard et al., 2016, Duvenaud et al., 2015]. They were inspired by the success of convolutional neural nets on computer vision tasks and have been applied in many domains, including analyzing large web graphs [Ying et al., 2018], designing printed circuit boards [Mirhoseini et al., 2021], and predicting chemical properties of molecules [Duvenaud et al., 2015, Gilmer et al., 2017]. An overview of GNNs is provided by Battaglia et al. [2018].

There has recently been growing interest in the application of machine learning techniques to solve CO problems such as the TSP and VRP [Bengio et al., 2021]. As many such problems have natural interpretations as graphs, a popular approach has been to use GNNs to solve them. A prominent early example is the work of Vinyals et al. [2015], who propose Pointer Networks and train them via supervised learning to solve TSP instances.

In CO problems generally, it is difficult to find a globally optimal solution but easier to compute a scalar quality metric for any given solution. As noted by Bengio et al. [2021], this makes RL, in which a system learns to maximize a scalar reward, a natural fit. Recent work [Dai et al., 2017, Kool et al., 2019, Lu et al., 2019, Sykora et al., 2020] has used RL to train GNN models and have attained impressive performance on the TSP, the VRP, and related problems.

The solutions from some neural methods come close to the quality of those from specialized TSP algorithms such as Concorde [Applegate et al., 2001], while requiring much less run-time to compute [Kool et al., 2019]. However, these methods all learn heuristics for constructing a single solution to a single problem instance. By the nature of NP-hard problems such heuristics will always be limited in the quality of their results; In this work, we show that a metaheuristic algorithm that searches over multiple solutions from the learned heuristic can offer better quality.

2.2 Optimization of Public Transit

The transportation optimization literature has extensively studied the Transit Network Design Problem. This problem is NP-complete [Quak, 2003], making it impractical to find optimal solutions for most cases. While analytical optimization and mathematical programming methods have been successful on small instances [van Nes, 2003, Guan et al., 2006], they struggle to realistically represent the problem [Guihaire and Hao, 2008, Kepaptsoglou and Karlaftis, 2009], and so metaheuristic approaches (as defined by Sörensen et al. [2018]) have been more widely applied. Historically, GAs, SA, and ant-colony optimization have been most popular, along with hybrids of these methods [Guihaire and Hao, 2008, Kepaptsoglou and Karlaftis, 2009]. But more recent work has adapted other metaheuristic algorithms such as BCO [Nikolić and Teodorović, 2013] and sequence-based selection hyper-heuristics [Ahmed et al., 2019], demonstrating that they outperform approaches based on GAs and SA.

On the other hand, while much work has used neural networks for predictive problems in urban mobility [Xiong and Schneider, 1992, Rodrigue, 1997, Chien et al., 2002, Jeong and Rilett, 2004, Çodur and Tortum, 2009, Li et al., 2020] and for other transit optimization problems such as scheduling and passenger flow control [Zou et al., 2006, Ai et al., 2022, Yan et al., 2023, Jiang et al., 2018], relatively little work has applied RL or neural networks (NNs) to the NDP. Darwish et al. [2020] and Yoo et al. [2023] both use RL to design a network and schedule for the Mandl benchmark [Mandl, 1980], a single small graph with just 15 nodes. Darwish et al. [2020] use a GNN approach inspired by Kool et al. [2019], while Yoo et al. [2023] uses tabular RL. Tabular RL approaches tend to scale poorly; meanwhile, in our own work we experimented with a nearly identical approach to Darwish et al. [2020], but found it did not scale beyond very small instances. Both these approaches also require a new model to be trained on each problem instance. The technique developed here, by contrast, is able to find good solutions for realistically-sized NDP instances of more than 100 nodes, and can be applied to problem instances unseen during training.

3 The Transit Network Design Problem

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

1. The route network $\mathcal{R}$ must be connected, allowing every node in $\mathcal{N}$ to be reached from every other node via transit.
2. The route network must contain exactly S routes, that is, $|\mathcal{R}| = S$, where S is a parameter set by the user.
3. Every route $r \in \mathcal{R}$ must be within stop limits $MIN \leq |r| \leq MAX$, where MIN and MAX are parameters set by the user.
4. No route $r \in \mathcal{R}$ may contain cycles; that is, it must include each node i at most once.

We here deal with the symmetric NDP, that is: $D = D^\top$, $(i, j, \tau_{ij}) \in \mathcal{E}_s$ iff. $(j, i, \tau_{ij}) \in \mathcal{E}_s$, and all routes are traversed both forwards and backwards by vehicles on them.

3.1 Markov Decision Process Formulation

A Markov Decision Process (MDP) is a formalism for describing a step-by-step problem-solving process, commonly used to define problems in RL. In an MDP, an **agent** interacts with an environment over a series of time steps. At each time step t, the environment is in some **state** $s_t \in \mathcal{S}$; the agent takes some **action** a_t which belongs to the set $\mathcal{A}_t$ of available actions in state s_t. This causes a transition to a new state $s_{t+1} \in \mathcal{S}$ according to the state transition distribution $P(s'|s, a)$, and also gives the agent a numerical **reward** $R_t \in \mathbb{R}$ according to the reward distribution $P(R|s, a, s')$. The agent acts according to a **policy** $\pi(a|s)$, which is a probability distribution over the available actions in each state. In RL, the goal is typically to **learn** a policy π through repeated interactions with the environment, such that π maximizes some measure of reward over time.

We here describe the MDP we use to represent the Transit Network Design Problem. At a high level, the MDP alternates at every step t between two modes: **extend**, where the agent selects an extension to the route r_t that it is currently planning; and **halt**, where the agent chooses whether to continue extending r_t or stop, adding it as-is to the transit network and beginning the planning of a new route. This alternation is captured by the state variable $\textbf{extend}_t \in \{\text{True}, \text{False}\}$, a boolean which changes its value after every step:

$$\textbf{extend}_t = \begin{cases} \neg\textbf{extend}_{t-1} & \text{if } t > 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{1}$$

More completely, the state s_t is composed of the city graph $\mathcal{C}$, the set of routes $\mathcal{R}_t$ planned so far, the state of the in-progress route r_t, and the mode variable $\textbf{extend}_t$. As $\mathcal{C}$ does not change with t, we represent the s_t as in eqn. 2.

$$s_t = (\mathcal{R}_t, r_t, \textbf{extend}_t) \tag{2}$$

The starting state is $s_0 = (\mathcal{R}_0 = \{\}, r_0 = [], \textbf{extend}_0 = \text{True})$.

When the expression ($\textbf{extend}_t = \text{True}$), the available actions are drawn from SP, the set of shortest paths between all node pairs. If $r_t = []$, then $\mathcal{A}_t = \{a | a \in \text{SP}, |a| \leq MAX\}$. Otherwise, $\mathcal{A}_t$ is comprised of paths $a \in \text{SP}$ satisfying all of the following conditions:

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

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

When ($\textbf{extend}_t$ = False), the action space is given by eqn. 3.

$$\mathcal{A}_t = \begin{cases} \{\text{continue}\} & \text{if} |r| < MIN \\ \{\text{halt}\} & \text{if} |r| = MAX \\ \{\text{continue}, \text{halt}\} & \text{otherwise} \end{cases} \tag{3}$$

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

Thus, the full state transition distribution is deterministic, and is described by eqn. 4.

$$s_t = \begin{cases} (\mathcal{R}_t = \mathcal{R}_{t-1}, r_t = \text{combine}(r_{t-1}, a_{t-1}), \text{False}) & \text{if } \textbf{extend}_{t-1} \\ (\mathcal{R}_t = \mathcal{R}_{t-1} \cup \{r_{t-1}\}, r_t = [], \text{True}) & \text{if } \neg\textbf{extend}_{t-1} \text{ and } a_{t-1} = \text{halt} \\ (\mathcal{R}_t = \mathcal{R}_{t-1}, r_t = r_{t-1}, \text{True}) & \text{if } \neg\textbf{extend}_{t-1} \text{ and } a_{t-1} = \text{continue} \end{cases} \tag{4}$$

When $|\mathcal{R}_t| = S$, the MDP terminates, giving the final reward $R_t = -C(\mathcal{C}, \mathcal{R}_t)$. The reward $R_t = 0$ at all prior steps.

This MDP formalization imposes some helpful biases on the solution space. First, it requires all transit routes to follow the street graph $(\mathcal{N}, \mathcal{E}_s)$; any route connecting i and j must also stop at all nodes along some path between i and j, thus biasing planned routes towards covering more nodes. Second, it biases routes towards being direct and efficient by forcing them to be composed of shortest paths; though in the limiting case a policy may construct arbitrarily indirect routes by choosing paths with length 2 at every step, this is unlikely as the majority of paths in SP are longer than two edges in realistic street graphs. Thirdly and finally, the alternation between deciding to whether to continue a route and deciding to how to extend it means that the probability of halting does not depend on how many different extensions are possible.

3.2 Cost Function

We can define the cost function in general as being composed of three components. The cost to riders is the average time of all passenger trips over the network:

$$C_p(\mathcal{C}, \mathcal{R}) = \frac{\sum_{i,j} D_{ij} \tau_{\mathcal{R}ij}}{\sum_{i,j} D_{ij}} \tag{5}$$

Where $\tau_{\mathcal{R}ij}$ is the time of the shortest transit trip from i to j given $\mathcal{R}$, including a time penalty p_T for each transfer. The operating cost is the total driving time of the routes:

$$C_o(\mathcal{C}, \mathcal{R}) = \sum_{r \in \mathcal{R}} \tau_r \tag{6}$$

Where τ_r is the time needed to completely traverse a route r in both directions.

To enforce the constraints on $\mathcal{R}$, we also add a term C_c, which is the fraction of node pairs that are not connected by $\mathcal{R}$ plus a measure of how much $|r| > MAX$ or $|r| < MIN$ across all routes. The cost function is then:

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

The weight $\alpha \in [0, 1]$ controls the trade-off between passenger and operator costs. 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 $\mathcal{C}$ and $\mathcal{R}$; this is done so that α will properly balance the two, and to stabilize training of the neural network policy. The values used are $w_p = (\max_{i,j} T_{ij})^{-1}$ and $w_o = (3S \max_{i,j} T_{ij})^{-1}$, where T is an $n \times n$ matrix of shortest-path driving times between every node pair.

4 Learned Planner

We propose to learn a policy $\pi_\theta(a|s)$ with parameters θ with the objective of maximizing $G = \sum_t R_t$ on the MDP described in section 3.1. Since reward is only given at the final timestep, we have:

$$G = -C(\mathcal{C}, \mathcal{R}_{final}) \tag{8}$$

By rolling out this policy on the MDP with some city $\mathcal{C}$, we can obtain a transit network $\mathcal{R}$ for that city. We denote this algorithm the Learned Planner (LP), or LP($\mathcal{C}, \alpha, \mathcal{R}_0 = \{\}$).

The policy π_θ is a neural network model parameterized by θ. Its "backbone" is a graph attention network [Brody et al., 2021] which treats the city as a fully-connected graph on the nodes $\mathcal{N}$, where each edge has an associated feature vector $\rceil_{ij}$ containing information about demand, existing transit connections, and the street edge (if one exists) between i and j. We note that a graph attention network operating on a fully-connected graph has close parallels to a Transformer model [Vaswani et al., 2017], but unlike Transformers this architecture enables the use of edge features that describe known relationships between elements.

The backbone GNN outputs node embeddings Y, which are operated on by one of two policy "heads", depending on the state s_t: NN_{ext} for choosing among extensions when $\textbf{extend}_t = \text{True}$, and NN_{halt} for deciding whether to halt when $\textbf{extend}_t = \text{False}$. The details of the network architecture are provided in Appendix B in the supplementary material.

4.1 Training

Following the work of Kool et al. [2019], we train the policy network using the policy gradient method REINFORCE with baseline [Williams, 1992] and set $\gamma = 1$. Since the reward R_t for the last step is the negative cost and at all other steps $R_t = 0$, by setting the discount rate $\gamma = 1$, the return G_t to each action a_t is simply $G_t = \sum_{t'} \gamma^{t'-t} R_t = -C(\mathcal{C}, \mathcal{R})$. The learning signal for each action a_t is $G_t - \text{baseline}(\mathcal{C}, \alpha)$, where the baseline function $\text{baseline}(\mathcal{C}, \alpha)$ is a separate Multi-Layer Perceptron (MLP) trained to predict the final reward obtained by the current policy for a given cost weight α and city $\mathcal{C}$.

The model is trained on a variety of synthetic cities and over a range of values of $\alpha \in [0, 1]$. S, n, MIN, and MAX are held constant during training. For each batch, a full rollout of the MDP episode is performed, the cost is computed, and back-propagation and weight updates are applied to both the policy network and the baseline network.

Each synthetic city begins construction by generating its nodes and street network using one of these processes chosen at random:

- Incoming 4-nn: Sample n random 2D points uniformly in a square to give $\mathcal{N}$. Add street edges to each node i from it's four nearest neighbours.
- Outgoing 4-nn: The same as the above, but add edges in the opposite direction.
- Voronoi: Sample m random 2D points, and compute their Voronoi diagram [Fortune, 1995]. Take the shared vertices and edges of the resulting Voronoi cells as $\mathcal{N}$ and $\mathcal{E}_s$. m is chosen so $|\mathcal{N}| = n$.
- 4-grid: Place n nodes in a rectangular grid as close to square as possible. Add edges from each node to its horizontal and vertical neighbours.
- 8-grid: The same as the above, but also add edges between diagonal neighbours.

For all models except Voronoi, each edge is then deleted with user-defined probability ρ. If the resulting street graph is not strongly connected (that is, all nodes are reachable from all other nodes), it is discarded and the process is repeated. Nodes are sampled in a 30km $\times$ 30km square, and a fixed vehicle speed of $v = 15$m/s is assumed to compute street edge weights $\tau_{ij} = ||(x_i, y_i) - (x_j, y_j)||_2 / v$. Finally, we generate the OD matrix D by setting diagonal demands $D_{ii} = 0$ and uniformly sampling off-diagonal elements $D_{ij} \sim [60, 800]$.

All neural network inputs are normalized so as to have unit variance and zero mean across the entire dataset during training. The scaling and shifting normalization parameters are saved as part of the model and applied to new data presented at test time.

5 Bee Colony Optimization

BCO is an algorithm inspired by how bees in a hive cooperate to search for nectar. At a high level, it works as follows. Given an initial problem solution $\mathcal{R}_0$ and a cost function C, a fixed number B of "bee" processes are initialized with $\mathcal{R}_b = \mathcal{R}_0 \; \forall \; b \in [0, B]$. Each bee makes a fixed number N_C of

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

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

random modifications to $\mathcal{R}_b$, discarding the modification if it increases cost $C(\mathcal{R}_b)$. Then each bee is randomly designated a "recruiter" or "follower", where $P(b = \text{follower}) \propto C(\mathcal{R}_b)$. Each follower bee b_f copies the solution of a random recruiter bee b_r, with probability inversely related to $C(\mathcal{R}_{b_r})$. These alternating steps of exploration and recruitment are repeated until some termination condition is met, and the lowest-cost solution $\mathcal{R}_{best}$ found over the process is returned.

In Nikolić and Teodorović [2013], BCO is adapted to the NDP by dividing the worker bees into two types, which apply different random modification processes. Given a network $\mathcal{R}_b$ with S routes for city $\mathcal{C}$ for each bee b, each bee selects a route $r \in \mathcal{R}_b$ with probability inversely related to the amount of demand r directly satisfies, and then selects a random terminal (first or last node) on r. Type-1 bees replace the chosen terminal with a random other terminal node in $\mathcal{N}$, and make the new route the shortest path between the new terminals. Meanwhile, type-2 bees choose with probability 0.2 to delete the chosen terminal from the route, and with probability 0.8 to add a random node neighbouring the chosen terminal to the route (at the start or end, depending on the terminal), making that node the new terminal. The overall best solution is updated after every N_P modification-and-recruitment steps (making one "iteration"), and the algorithm performs I iterations before halting. Henceforth, "BCO" refers specifically to this NDP algorithm.

We propose a modification of this algorithm, called Neural BCO ("NBCO" henceforth), in which the type-1 bees are replaced by "neural bees". A neural bee selects a route $r \in \mathcal{R}$ for modification in the same manner as the type-1 and type-2 bees, but instead of selecting a terminal on r, it rolls out our learned policy π_θ to replace r with a new route $r' \leftarrow \text{LP}(\mathcal{C}, \alpha, \mathcal{R} \setminus \{r\})$. We replace the type-1 bee because its action space (replacing one route by a shortest path) is a subset of the action space of the neural bee (replacing one route by a new route composed of shortest paths), while the type-2 bee's action space is quite different. The algorithm is otherwise unchanged; for the full details, we refer the reader to Nikolić and Teodorović [2013].

6 Experiments

In all experiments, the policies π_θ used are trained on a dataset of $2^{15} = 32,768$ synthetic cities with $n = 20$. A 90:10 training:validation split of this dataset is used; after each epoch of training, the model is evaluated on the validation set, and at the end of training, the model weights from the epoch with the best validation-set performance are returned. Data augmentation is applied each time a city is used for training. This consists of multiplying the node positions (x_i, y_i) and travel times τ_{ij} by a random factor $c_s \sim [0.4, 1.6]$, rotating the node positions about their centroid by a random angle $\phi \sim [0°, 360°)$, and multiplying D by a random factor $c_d \in [0.8, 1.2]$. During training and evaluation, constant values $S = 10, MIN = 2, MAX = 15$ are used. Training proceeds for 5 epochs, with a batch size of 64 cities. When training with different random seeds, the dataset is held constant across seeds but the data augmentation is not.

All evaluations are performed on the Mandl [Mandl, 1980] and Mumford [Mumford, 2013a] city datasets, two popular benchmarks for evaluating NDP algorithms [Mumford, 2013b, John et al., 2014, Kılıç and Gök, 2014, Ahmed et al., 2019]. The Mandl dataset is one small synthetic city, while the Mumford dataset consists of four synthetic cities, labelled Mumford0 through Mumford3, that range in size from $n = 30$ to $n = 127$, and gives values of S, MIN, and MAX to use when benchmarking on each city. The values n, S, MIN, and MAX for Mumford1, Mumford2, and Mumford3 are taken from three different real-world cities and their existing transit networks, giving the dataset a degree of realism. Details of these benchmarks are given in Table 1.

For both BCO and NBCO, we set all algorithmic parameters to the values used in the experiments of Nikolić and Teodorović [2013]: $B = 10, N_C = 2, N_P = 5, I = 400$. We also ran BCO for up to $I = 2,000$ on several cities, but found this did not yield any improvement over $I = 400$. We run NBCO with equal numbers of neural bees and type-2 bees, just as BCO uses equal numbers of type-1 and type-2 bees. Hyperparameter settings of the policy's model architecture and training process were arrived at by a limited manual search; for their values, we direct the reader to the configuration files contained in our code release. We set the constraint penalty weight $\beta = 5$ in all experiments.

6.1 Results

We compare LP, BCO, and NBCO on Mandl and the four Mumford cities. To evaluate LP, we perform 100 rollouts and choose the lowest-cost $\mathcal{R}$ from among them (denoted LP-100). Each algorithm is run across a range of 10 random seeds, with a separate policy network trained with that seed. We report results averaged over all of the seeds. Our main results are summarized in Table 2, which shows results at three different α values, $0.0, 1.0$, and 0.5, which optimize for the operators' perspective, the passengers' perspective, and a balance of the two. This table also contains results for two ablation experiments: one in which LP was rolled out $40,000$ times instead of 100 (denoted LP-40k), and one in which we ran a variant of NBCO with only neural bees, no type-2 bees.

The results show that while BCO performs best on the two smallest cities in most cases, its relative performance worsens considerably when $n = 70$ or more. On Mumford1, 2, and 3, for each α, LP matches or outperforms BCO. Meanwhile, NBCO with a mixture of bee types performs best overall on these three cities. It is better than LP-100 in every instance, improving on its cost by about 6% in most cases at $\alpha = 1.0$ and 0.5, and by up to 20% at $\alpha = 0.0$; and it improves on BCO by 33% to 53% on Mumford3 depending on α.

NBCO does fail to obey route length limits on 1 out of 10 seeds when $\alpha = 0.0$. This may be due to $\alpha = 0.0$ causing the benefits from under-length routes overwhelm the cost penalty due to a few routes being too long. This could likely be resolved by simply increasing β or adjusting the specific form of C_c.

Table 2: Average final cost $C(\mathcal{C}, \mathcal{R}, \alpha)$ achieved by each method over 10 random seeds, for three different settings of cost weight α. Bold indicates the best in each column. Orange indicates that one seed's solution violated a constraint, red indicates two or three seeds did so. Percentages are standard deviations over the 10 seeds.

City Method	Mandl	Mumford0	Mumford1	Mumford2	Mumford3
			$\alpha = 0.0$		
BCO	**0.276 ± 17%**	**0.272 ± 16%**	0.854 ± 32%	0.692 ± 40%	0.853 ± 35%
LP-100	0.317 ± 25%	0.487 ± 66%	0.853 ± 43%	0.688 ± 26%	0.710 ± 24%
LP-40k	0.273 ± 26%	0.440 ± 68%	0.805 ± 41%	0.665 ± 27%	0.690 ± 25%
NBCO	0.279 ± 20%	0.298 ± 41%	**0.623 ± 26%**	**0.537 ± 44%**	**0.572 ± 46%**
No-2-NB	0.290 ± 18%	0.295 ± 41%	0.670 ± 53%	0.605 ± 48%	0.574 ± 49%
			$\alpha = 0.5$		
BCO	**0.328 ± 3%**	**0.563 ± 2%**	1.015 ± 42%	0.710 ± 36%	0.944 ± 29%
LP-100	0.343 ± 10%	0.638 ± 22%	0.742 ± 24%	0.617 ± 14%	0.612 ± 13%
LP-40k	0.329 ± 9%	0.619 ± 20%	0.718 ± 22%	0.606 ± 14%	0.601 ± 14%
NBCO	0.331 ± 7%	0.571 ± 6%	**0.627 ± 8%**	**0.532 ± 5%**	**0.584 ± 37%**
No-2-NB	0.330 ± 4%	0.588 ± 7%	0.639 ± 15%	0.589 ± 34%	0.596 ± 37%
			$\alpha = 1.0$		
BCO	**0.314 ± 1%**	0.645 ± 5%	0.739 ± 37%	0.656 ± 38%	1.004 ± 40%
LP-100	0.335 ± 2%	0.738 ± 6%	0.600 ± 3%	0.534 ± 2%	0.504 ± 1%
LP-40k	0.325 ± 1%	0.709 ± 5%	0.587 ± 3%	0.528 ± 2%	0.498 ± 1%
NBCO	0.317 ± 1%	**0.637 ± 6%**	**0.564 ± 3%**	**0.507 ± 1%**	**0.481 ± 2%**
No-2-NB	0.320 ± 1%	0.668 ± 7%	0.570 ± 2%	0.511 ± 2%	0.486 ± 2%

6.2 Ablations

We first observe that under the parameter settings used here, BCO and NBCO both consider a total of $B \times N_C \times N_P \times I = 40,000$ different networks over a single run. To see whether NBCO's improvement over LP-100 is simply due to its exploring more solutions, we performed the LP-40k experiments, taking $40,000$ samples from LP instead of 100. The results for LP-40k in Table 2 show that it while it improves on NBCO on Mandl, for all larger cities it is only slightly better than LP-100. The gap between LP-40k and NBCO is at least 54% larger than the gap between LP-40k and LP-100 on each Mumford city for each α value, and in 10 of the 12 cases it is more than twice as large. This indicates that the main factor in NBCO's improvement over LP is the metaheuristic algorithm that guides the search through solution space.

To examine the importance of the type-2 bee to NBCO, we run another set of experiments with a variant of NBCO with no type-2 bees, only neural bees, denoted No-2-NB. Again the results are displayed in Table 2. We observe that with the exception of Mumford0 with $\alpha = 0.0$ and Mandl with $\alpha = 0.5$, its performance is worse than with both types of bees: the very different action space of the type-2 bees is a useful complement to the neural bees. However, this variant still outperforms BCO and both LP variants for most cities and α values: it is the guidance of the learned heuristic by the bee-colony metaheuristic that is responsible for most of NBCO's superior performance.

6.3 Trade-offs Between Passenger and Operator Costs

There is a necessary trade-off between minimizing the passenger cost C_p and the operator cost C_o: making transit routes longer increases C_o, but allows more and faster direct connections between stops, and so may decrease C_p. The weight α can be set by the user to indicate how much they care about C_p versus C_o, and each algorithm output will change accordingly. Figure 1 illustrates the trade-offs made by the different methods, as we vary α over its full range $[0, 1]$ in steps of 0.1, except for LP-40k, for which we only plot values for $\alpha = 0.0, 0.5$, and 1.0. For the two smallest cities (sub-figures 1a and 1b), BCO offers a superior trade-off for most α, but for the larger cities Mumford1, 2, and 3, NBCO's solutions not only dominate those of BCO, they achieve a much wider range of C_p and C_o than either of BCO or LP, which will be more satisfactory if the user cares only about one or the other component.

Both LP and BCO have more narrow ranges of C_p and C_o on the three larger cities, but the ranges are mostly non-overlapping. Some of NBCO's greater range seems to be due to combining the non-overlapping ranges of the constituent parts, but NBCO's range is greater than the union of LP's and BCO's ranges. This implies that the larger action space of the neural bee versus the type-1 bee allows NBCO to explore a much wider range of solutions by taking wider "steps" in solution space. Meanwhile, LP-40k has about the same range as LP-100, implying that these wide steps must be guided by the metaheuristic to eventually reach a wider space of solutions.

7 Discussion

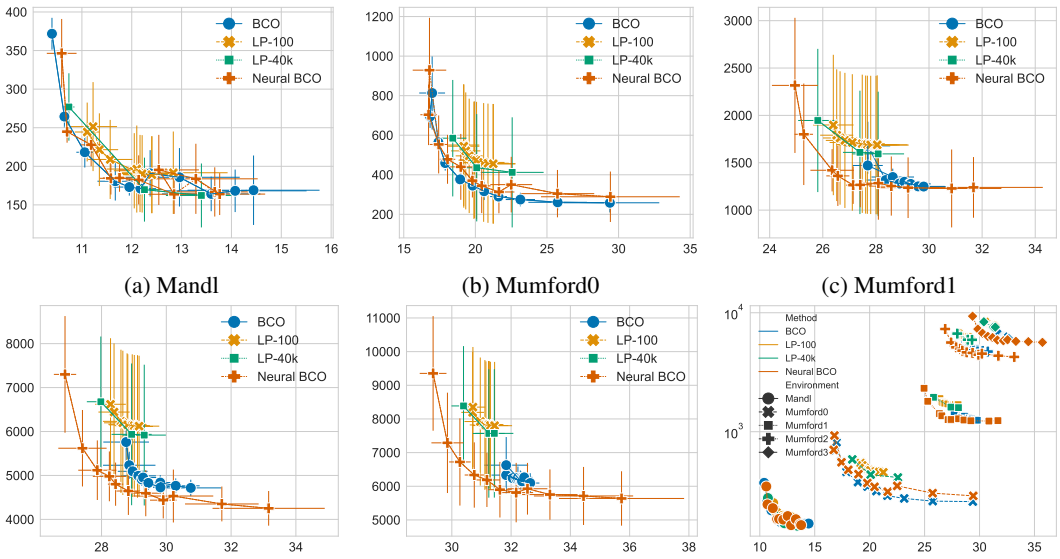
7.1 Limitations

Bee Colony Optimization is just one instance of the broad class of metaheuristic algorithms, and while the results we present are promising, it remains to be seen whether incorporating learned heuristics into metaheuristic algorithms is a sound algorithmic strategy in general. Furthermore, in this work we consider such a combination in light of only one CO problem, the NDP. This is a very interesting and impactful problem, but evaluating this method on a wider variety of CO problems such as the TSP and VRP would more broadly establish the usefulness of this strategy.

We also note that, while the Mumford dataset is a widely used benchmark, it is still synthetic data. Establishing whether our method would be useful for transit planning in real-world cities will require evaluating on a real-world dataset.

7.2 Conclusions

Ultimately, it is doubtful whether a single-pass generation heuristic like that implemented by the GNN will be capable of outperforming search based methods like metaheuristics on combinatorial



(a) Mandl (b) Mumford0 (c) Mumford1

(d) Mumford2 (e) Mumford3 (f) All cities

Figure 1: Trade-offs achieved by different methods between passenger cost C_p (on the x-axis) and operator cost C_o (on the y-axis), across values of α evenly spaced over the range $[0, 1]$, averaged over 10 random seeds. Both axes are in units of minutes. We wish to minimize both values, so the lower-left direction in each plot represents improvement. The y-axis of sub-figure 1f is log-scaled to better fit all curves without flattening them, while the rest are linear. A line links two points if they have adjacent α values, so these curves show a smooth progression from low C_o to low C_p as α increases.

optimization problems like the NDP. By these problems' nature, there is no one-step algorithm for finding optimal solutions, and any fast-to-compute heuristic will necessarily be approximate. Consequently, methods for exploring the solution space on a given instance have a general advantage over such heuristics. But we have shown that the choice of heuristics can have a significant impact on the quality of the solutions found by a metaheuristic, and learned heuristics in particular can significantly benefit metaheuristic algorithms when used as some of their sub-heuristics.

In terms of the applicability of these methods in real cities, we note that both LP and Neural BCO outperform BCO on all three cities - Mumford1, 2, and 3 - that were designed to match a specific real-world city in scale. Furthermore, the gap between BCO and the other methods grows with the size of the city. This suggests that Neural BCO may scale better to much larger problem sizes - which is significant, as some real-world cities have hundreds or even thousands of bus stop locations [Société de transport de Montréal, 2013].

We note that better results could likely be achieved by training a policy directly in a metaheuristic context, rather than training it in isolation and then applying it in a metaheuristic as was done here. It would also be interesting to use multiple separately-trained models as different heuristics within a metaheuristic algorithm, as opposed to the single model used in our experiments. This could be seen as a form of ensemble method, with the metaheuristic intelligently combining the strengths of the different learned models to get the best use out of each.

We would also like to explore the training of a further Machine Learning (ML) component to act as the higher-level metaheuristic, creating an entirely learned method for searching the solution space for particular problem instances. Recent work on few-shot adaptation in RL [Behbahani et al., 2023] may provide a promising starting point.

Beyond studying the relative performance of machine learning and metaheuristic approaches and their combination, we hope by this work to draw the attention of the machine community to the NDP. It is a uniquely challenging combinatorial optimization problem with real-world impact, with much potential for novel and useful study by our discipline.

References

Leena Ahmed, Christine Mumford, and Ahmed Kheiri. Solving urban transit route design problem using selection hyper-heuristics. *European Journal of Operational Research*, 274(2):545–559, 2019.

Guanqun Ai, Xingquan Zuo, Gang Chen, and Binglin Wu. Deep reinforcement learning based dynamic optimization of bus timetable. *Applied Soft Computing*, 131:109752, 2022.

David Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. Concorde tsp solver. https://www.math.uwaterloo.ca/tsp/concorde/index.html, 2001.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Feryal Behbahani, Jakob Bauer, Kate Baumli, Satinder Baveja, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2021. URL `https://arxiv.org/abs/2105.14491`.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Steven I-Jy Chien, Yuqing Ding, and Chienhung Wei. Dynamic bus arrival time prediction with artificial neural networks. *Journal of transportation engineering*, 128(5):429–438, 2002.

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

Ahmed Darwish, Momen Khalil, and Karim Badawi. Optimising public bus transit networks using deep reinforcement learning. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, abs/1606.09375, 2016. URL `http://arxiv.org/abs/1606.09375`.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

Steven Fortune. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, pages 225–265, 1995.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/gilmer17a.html`.

J.F. Guan, Hai Yang, and S.C. Wirasinghe. Simultaneous optimization of transit line configuration and passenger line assignment. *Transportation Research Part B: Methodological*, 40:885–902, 12 2006. doi: 10.1016/j.trb.2005.12.003.

Valérie Guihaire and Jin-Kao Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008.

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

Zhibin Jiang, Wei Fan, Wei Liu, Bingqin Zhu, and Jinjing Gu. Reinforcement learning approach for coordinated passenger inflow control of urban rail transit in peak hours. *Transportation Research Part C: Emerging Technologies*, 88:1–16, 2018. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2018.01.008. URL https://www.sciencedirect.com/science/article/pii/S0968090X18300111.

Matthew P. John, Christine L. Mumford, and Rhyd Lewis. An improved multi-objective algorithm for the urban transit routing problem. In Christian Blum and Gabriela Ochoa, editors, *Evolutionary Computation in Combinatorial Optimisation*, pages 49–60, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44320-0.

Konstantinos Kepaptsoglou and Matthew Karlaftis. Transit route network design problem: Review. *Journal of Transportation Engineering*, 135(8):491–505, 2009. doi: 10.1061/(ASCE)0733-947X(2009)135:8(491).

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

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

Fatih Kılıç and Mustafa Gök. A demand based route generation algorithm for public transit network design. *Computers & Operations Research*, 51:21–29, 2014. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2014.05.001. URL https://www.sciencedirect.com/science/article/pii/S0305054814001300.

Can Li, Lei Bai, Wei Liu, Lina Yao, and S Travis Waller. Graph neural network for robust public transit demand prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2019.

Christoph E Mandl. Evaluation and optimization of urban public transportation networks. *European Journal of Operational Research*, 5(6):396–404, 1980.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Christine L Mumford. Download link to the mumford dataset. https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp.zip, 2013a. Accessed: 2023-03-24.

Christine L Mumford. New heuristic and evolutionary operators for the multi-objective urban transit routing problem. In *2013 IEEE congress on evolutionary computation*, pages 939–946. IEEE, 2013b.

Miloš Nikolić and Dušan Teodorović. Transit network design by bee colony optimization. *Expert Systems with Applications*, 40(15):5945–5955, 2013.

CB Quak. Bus line planning. *A passenger-oriented approach of the construction of a global line network and an efficient timetable. Master's thesis, Delft University, Delft, Netherlands*, 2003.

Jean-Paul Rodrigue. Parallel modelling and neural networks: An overview for transportation/land use systems. *Transportation Research Part C: Emerging Technologies*, 5(5):259–271, 1997. ISSN 0968-090X. doi: https://doi.org/10.1016/S0968-090X(97)00014-4. URL https://www.sciencedirect.com/science/article/pii/S0968090X97000144.

Société de transport de Montréal. Everything about the stm, 2013. URL https://web.archive.org/web/20130610123159/http://www.stm.info/english/en-bref/a-toutsurlaSTM.htm. Accessed: 2023-05-17.

Kenneth Sörensen, Marc Sevaux, and Fred Glover. A history of metaheuristics. In *Handbook of heuristics*, pages 791–808. Springer, 2018.

Quinlan Sykora, Mengye Ren, and Raquel Urtasun. Multi-agent routing value iteration network. In *International Conference on Machine Learning*, pages 9300–9310. PMLR, 2020.

Rob van Nes. Multiuser-class urban transit network design. *Transportation Research Record*, 1835 (1):25–33, 2003. doi: 10.3141/1835-04. URL `https://doi.org/10.3141/1835-04`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Yihua Xiong and Jerry B Schneider. Transportation network design using a cumulative genetic algorithm and neural network. *Transportation Research Record*, 1364, 1992.

Haoyang Yan, Zhiyong Cui, Xinqiang Chen, and Xiaolei Ma. Distributed multiagent deep reinforcement learning for multiline dynamic bus timetable optimization. *IEEE Transactions on Industrial Informatics*, 19:469–479, 2023.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *CoRR*, abs/1806.01973, 2018. URL `http://arxiv.org/abs/1806.01973`.

Sunhyung Yoo, Jinwoo Brian Lee, and Hoon Han. A reinforcement learning approach for bus network design and frequency setting optimisation. *Public Transport*, pages 1–32, 2023.

Liang Zou, Jian-min Xu, and Ling-xiang Zhu. Light rail intelligent dispatching system based on reinforcement learning. In *2006 International Conference on Machine Learning and Cybernetics*, pages 2493–2496, 2006. doi: 10.1109/ICMLC.2006.258785.

Muhammed Yasin Çodur and Ahmet Tortum. An artificial intelligent approach to traffic accident estimation: Model development and application. *Transport*, 24(2):135–142, 2009. doi: 10.3846/1648-4142.2009.24.135-142.

A Symbols

This paper makes use of a large number of symbols. Tables 3 and 4 list and defines all of these in one place for ease of reference. The symbols are presented in alphabetical order, with greek following latin. The listing is broken up into two tables so that each will fit on one page.

B Neural Network Policy

In this section we will describe in detail the architecture of the neural network policy π_θ that drives the Learned Planner algorithm.

B.1 Input Features

Each node i in the graph has a feature vector $\mathbf{x}_i$ comprised of the following components:

- (x_i, y_i), the spatial coordinates of i
- δ_{si}, the number of street edges connected to i
- δ_{Di}, the number of nodes with any demand to/from i

- $\sum_{j \in \mathcal{N}} D_{ij}$, the total demand to/from i

Let $X = \mathbf{x}_i \forall i \in \mathcal{N}$, a tensor of all node feature vectors.

A city with some (possibly incomplete) transit network can be viewed as a fully connected graph with edge features that indicate the presence or absence of streets, driving times, transit connections, and levels of demand between each pair of nodes. Our policy network is a GNN that operates on this fully connected graph. The feature vector $\mathbf{e}_{ij}$ associated with each edge (i, j) is composed of the following data:

- D_{ij}, the demand between the nodes
- T_{ij}, the shortest-path driving time between the nodes
- $s_{ij} = 1$ if $(i, j, \tau_{ij}) \in \mathcal{E}_s$, 0 otherwise
- τ_{ij} if $s_{ij} = 1$, 0 otherwise
- $c_{ij} = 1$ if $\mathcal{R}$ links i to j, 0 otherwise
- $c_{0ij} = 1$ if j can be reached from i over $\mathcal{R}$ with no transfers, 0 otherwise
- $c_{1ij} = 1$ if one transfer is needed to reach j from i over $\mathcal{R}$, 0 otherwise
- $c_{2ij} = 1$ if two transfers are needed to reach j from i over $\mathcal{R}$, 0 otherwise
- $\tau_{\mathcal{R}ij}$ if $c_{ij} = 1$, 0 otherwise
- self $= (i = j)$, a binary feature indicating whether this edge connects a node to itself

Finally, a vector $\mathbf{s}_t$ of overall state features is also used by the policy network, composed of these elements:

- $|\mathcal{R}_t|$, the number of completed routes so far
- $S - |\mathcal{R}_t|$, the number of routes left to plan
- α, the parameter of the cost function C being applied in this instance
- $\sum_{r \in \mathcal{R}_t} \tau_r$, the total time of all completed routes so far, which reflects the cost of operation

B.2 Architecture

The policy π_θ is a neural network with three components: a GNN "backbone", and two policy heads: a halting module NN_{halt} and an extension module NN_{ext}. All nonlinearities are ReLU functions. All attention modules are multi-headed with 4 heads. A common embedding dimension of $d_{embed} = 64$ is used; unless otherwise specified, each component of the system outputs vectors of this dimension.

B.2.1 Backbone

The backbone is a graph attention network composed of five GATv2 layers separated by nonlinearities. Each layer produces node descriptors, a nonlinearity is applied to these, and they are passed as input to the next layer. The network takes as input the node feature collection X and the edge feature E, and the final layer outputs a collection of node embeddings $Y = \{\mathbf{y}_i \ \forall i \ \in \mathcal{N}\}$. These node embeddings are used by the two policy heads to compute action probabilities.

B.2.2 Halting module

The halting module NN_{halt} first applies a two-layer multi-head attention module to compute a descriptor of the in-progress route r_t. The first layer takes the mean of the node embeddings $\mathbf{y}_{mean} = \frac{\sum_{i \in} \mathbf{y}_i}{n}$ as the query vector, and $[y_i | i \in r_t]$ as the key and value sequence; the second layer takes the output of the first as the query, and uses the same key and value sequence, outputting a route embedding vector $\mathbf{r}_t$. This is concatenated with $\mathbf{s}_t$ and τ_{r_t} to give a vector $\mathbf{r'}_t$, and an MLP is applied to $\mathbf{r'}_t$, outputting a scalar h. We apply a sigmoid to h get the halting policy:

$$\pi(\text{halt}) = \sigma(h), \ \pi(\text{continue}) = 1 - \sigma(h) \tag{9}$$

The MLP has 1 hidden layer with dimension $d_{embed} * 2$.

B.2.3 Extension module

The extension module NN_{ext} is finely tuned to the structure of the cost function. Considering first the case where $\alpha = 1$, we observe that the quality of any candidate route r is solely a function of the edges (i, j, τ_{rij}) that it adds to the set $\mathcal{E}_{\mathcal{R}}$ of direct-transit-connection edges, where τ_{rij} denotes the time to get from i to j on route r. Therefore, for each path a we consider when extending r to get r', a reasonable heuristic is to compute a scalar score o_{ij} for each edge $(i, j, \tau_{r'ij})$, and let the path's "quality" be their sum $o_a = \sum_{i,j \in a} o_{ij}$. Following this intuition, for each node pair, we concatenate $[\mathbf{y}_i, \mathbf{y}_j, \mathbf{e}_{ij}, \tau_{r'ij}]$ into a vector and apply an MLP to to compute the scalar o_{ij}. We then set $o_{ij} \leftarrow 0$ if $c_{0ij} = 1$, since directly-connected node-pairs are unlikely to change the quality of the solution if connected by an additional route. Finally, we sum these to get o_a.

If $\alpha > 0$, then the quality of r still depends on the edges it adds to $\mathcal{E}_{\mathcal{R}}$, but also on its length, and whether it is a good choice to extend r by a depends generally on the state $\mathcal{S}$. Thus, we concatenate $[o_a, \tau_a, \mathbf{s}_t]$ and apply another MLP to obtain a final score $\hat{o}_a$ for each candidate path. The extension policy is then the softmax of these values:

$$\pi(p) = \frac{e^{\hat{o}_a}}{\sum_{a' \in \mathcal{A}} e^{\hat{o}_{a'}}} \tag{10}$$

All of NN_{ext}'s constituent MLPs have 2 hidden layers, each with dimension 8. This low dimensionality considerably aids speed, as these MLP must be applied serially many times over a rollout of the policy.

B.3 Baseline Network

The baseline function used to compute the learning signal $G_t - b(s_t)$ during training is an MLP with 2 hidden layers of dimension 36. As the return G_t is the same for all t by construction, the baseline need only depend on the details of the problem instance, namely α and $\mathcal{C}$. The input to $b(s_t)$ is a vector composed of α and several statistics of $\mathcal{C}$: the average of the node features $\frac{\sum_i \mathbf{x}_i}{n}$, the total demand $\sum_{i,j} D_{ij}$, and the means and standard deviations of the elements of D and T.

B.4 Training Hyper-Parameters

Training was performed using the well-known Adam optimizer. During training, dropout was applied between all layers of all components of the policy network, but was not applied to the baseline network. At the start of training, a calibration phase is carried out, in which the randomly initialized policy is run over the entire training dataset without computing gradients or updating its weights. During this phase, two things happen:

- The mean and standard deviation of all of the policy's inputs are computed, and these are used to normalize the network's inputs during training and afterwards.
- The baseline network is trained to predict $G_t = -C(\mathcal{C}, \text{LP}(\mathcal{C}, \alpha, \{\}))$

The pre-training of the baseline networks allows it to provide an accurate baseline as soon as training of the policy begins, which we found helps to stabilize training, as does the input normalization.

Table 5 gives the hyper-parameters used during training.

Table 3: Symbols (Part 1)

Symbol	Definition
a	A path in SP.
a_t	The action chosen in the MDP at step t. This will be "halt" or "continue" if $\mathbf{extend}_t =$ False, or a path in SP if $\mathbf{extend}_t =$ True.
b	Denotes a single bee in BCO and Neural BCO.
B	A parameter of the BCO and Neural BCO algorithms: the number of bees used.
C	A cost function $C : \mathcal{C}, \mathcal{R} \to \mathbb{R}^+$ for a set of transit routes $\mathcal{R}$ operating on a city $\mathcal{C}$.
C_c	The cost of constraint violations of a transit network $\mathcal{R}$.
C_o	The cost to the operator of a transit network $\mathcal{R}$.
C_p	The cost to passengers of a transit network $\mathcal{R}$.
$\mathcal{C}$	$\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D)$, an augmented graph representing a city.
D	$n \times n$ OD matrix giving the travel demand (in number of trips) between every node pair $(i, j) \in \mathcal{N} \times \mathcal{N}$.
$\mathbf{e}_{ij}$	A feature vector of a node pair i, j.
E	A collection of feature vectors for all node pairs $i, j \in \mathcal{N} \times \mathcal{N}$.
$\mathbf{extend}_t$	A boolean variable indicating whether MDP step t is a route-extending step (True) or a step where it decides whether to halt (False).
$\mathcal{E}_\mathcal{R}$	A set of weighted edges (i, j, τ_{rij}) for every pair of nodes (i, j) that are directly connected by some route $r \in \mathcal{R}$.
$\mathcal{E}_s$	A set of weighted edges (i, j, τ_{ij}) representing streets between nodes in $\mathcal{N}$, where τ_{ij} is the time needed to drive from i to j along the connecting street.
G_t	The cumulative return received by an MDP agent from step t onwards.
i	Denotes a single node in $\mathcal{N}$.
I	The number of iterations performed by BCO and Neural BCO before terminating.
j	Denotes a node in $\mathcal{N}$ distinct from i.
m	The number of initial points used when randomly generating a Voronoi-style street network.
MIN	A lower limit on the number of stops on any route.
MAX	An upper limit on the number of stops on any route.
n	The number of nodes in a city graph.
N_C	The number of search moves made by each bee in BCO and Neural BCO between recruitment steps.
N_P	The number of modification-and-recruitment steps per iteration of BCO and NeuralBCO
NN_{ext}	A neural network component of π, used to compute the probability of each extension action.
NN_{halt}	A neural network component of π, used to compute the probability of the "halt" action.
$\mathcal{N}$	A set of candidate transit stop locations, where $n \equiv \lvert\mathcal{N}\rvert$.
o_a	A scalar "score" for a path computed at an intermediate stage of the learned planner, computed by summing over o_{ij} for all $i, j \in a$.
o_{ij}	A scalar "score" for a pair of nodes computed at an intermediate stage of the learned planner.
p_T	The time taken by a passenger to make a transfer between two transit lines, assumed to be constant across all passengers and all lines.
r	A transit route, defined as a sequence of stops $[i, j, ..., k]$ in $\mathcal{N}$.
r_t	The partial transit route being built at step t of the MDP.
R_t	The reward received by an MDP agent at step t.

Table 4: Symbols (Part 2)

Symbol	Definition
$\mathcal{R}$	A set of transit routes composing a transit network.
$\mathcal{R}_b$	The transit network solution belonging to bee b in BCO and Neural BCO.
$\mathcal{R}_t$	The partial transit network at step t of the MDP.
s_t	The state $(\mathcal{C}, \mathcal{R}_t, r_t, \mathbf{extend}_t)$ of the route-planning MDP at step t.
$\mathbf{s}_t$	A vector describing some features of the global MDP state at step t.
S	A user-defined value giving the number of routes that a complete transit network should contain.
SP	The set of shortest paths over $\mathcal{E}_s$ between every node pair $(i, j) \in \mathcal{N} \times \mathcal{N}$.
t	The timestep index of the route-planning MDP.
T	An $n \times n$ matrix of drive times along the shortest paths through $\mathcal{E}_s$ for all node pairs $(i, j) \in \mathcal{N} \times \mathcal{N}$.
v	The assumed speed of all transit vehicles.
w_o	A scaling weight applied to C_o in the cost function C.
w_p	A scaling weight applied to C_p in the cost function C.
X	A collection of feature vectors for all nodes $i \in \mathcal{N}$.
$\mathbf{x}_i$	A feature vector of a node i.
Y	A collection of vector embeddings produced by the policy's GNN for all nodes $i \in \mathcal{N}$.
(x_i, y_i)	A point in the 2D plane defined for each node $i \in \mathcal{N}$.
α	A weight that controls the tradeoff between C_p and C_o in the cost function C.
β	A weight applied to C_c in the cost function C.
π	A policy for the MDP.
τ_r	The time needed to completely traverse transit route r in both directions.
τ_{rij}	The time needed to travel from stop i to stop j on route r, if r directly connects those stops.
$\tau_{\mathcal{R}ij}$	The time of the shortest possible trip through the transit network $\mathcal{R}$ from stop i to stop j, which may require transfers between routes.

Table 5: 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}
Dropout rate	0.23
S	10
MIN	2
MAX	12