Augmenting Transit Network Design Algorithms with Deep Learning

Andrew Holliday[1] and Gregory Dudek

***Abstract*— This paper considers the use of deep learning models to enhance optimization algorithms for transit network design. Transit network design is the problem of determining routes for transit vehicles that minimize travel time and operating costs, while achieving full service coverage. State-of-the-art meta-heuristic search algorithms give good results on this problem, but can be very time-consuming. In contrast, neural networks can learn sub-optimal but fast-to-compute heuristics based on large amounts of data. Combining these approaches, we develop a fast graph neural network model for transit planning, and use it to initialize state-of-the-art search algorithms. We show that this combination can improve the results of these algorithms on a variety of metrics by up to 17%, without increasing their run time; or they can match the quality of the original algorithms while reducing the computing time by up to a factor of 50.**

I. Introduction

The design of urban transit networks is an important problem in urban planning. The layout of a transit network can have a major impact on its quality, and consequently on its level of use. Consequently, this problem - the Transit Network Design Problem (NDP) - has received much attention in the literature. Most of this work brings approaches from classical and stochastic optimization to bear on the problem, and the most successful approaches to-date have been meta-heuristic algorithms such as Genetic Algorithms and Simulated Annealing. By comparison, little cross-over exists between the literature on this problem and that on neural networks [1], [2], [3].

As noted by one prominent researcher in transit optimization [4], meta-heuristic algorithms that search a solution space depend on three aspects for their success:

1) The representation of solutions,
2) The algorithm for constructing the initial solution,
3) The "moves" in solution space chosen at each step of search.

Regardless of the algorithm, good performance will depend on these three aspects. In this work, we address the second aspect.

We propose a novel approach that uses machine learning with deep neural networks, known as deep learning, to initialize solution improvement methods. Specifically, we train a graph neural network (GNN) model on a synthetic dataset to produce transit networks that minimize a cost function, and then we apply the GNN model to new cities, unseen during training, to generate transit networks for those cities. We then use these transit networks as the initial solutions in two recently-proposed meta-heuristic algorithms. We find that this improves the algorithms' results, especially on the largest and most challenging problem instances. To aid in replication and extension of this work, we have made our code and data publicly available [2].

The rest of the paper proceeds as follows. Section II surveys related work on transit optimization and on Deep Reinforcement Learning (DRL) for optimization. Section III provides a formal specification of the NDP. Section IV provides the details of our algorithm. Section V describes our experiments and their results, comparing our approach with existing approaches, and Section VI summarizes our conclusions and presents possible directions for further research.

II. Related Work

A. Transit Optimization

The design of a mass transit system is comprised of three sub-problems:

- The Transit Network Design Problem (NDP): laying out transit routes and stops,
- The Frequency-Setting Problem (FSP): determining the frequency of vehicle departures on each route,
- The Timetabling Problem (TP): assigning vehicles, and possibly drivers, to routes and departure times.

In this work, we consider only the NDP. The NDP is a combinatorial optimization (CO) problem that has been the subject of much study in the transportation optimization literature. It is NP-complete [5], so finding a globally optimal solution is infeasible in most instances of the problem.

Analytical optimization and mathematical programming methods have been applied to small NDP instances with some success [6], [7]. But as noted both by [1] and [2] in their reviews of the subject, these methods are inflexible and struggle to realistically represent the problem. Consequently, heuristic and meta-heuristic approaches, as defined by [8], have been more widely used. Of these, Genetic Algorithms (GAs), Simulated Annealing (SA), and Ant Colony Optimization have most frequently been applied to the NDP, along with hybrids of these methods [1], [2]. More recently, [9] have proposed an algorithm based on the Bee Colony Optimization (BCO) meta-heuristic, and [10] used a sequence-based selection hyper-heuristic. Both methods were shown to outperform previous approaches based on GAs and SA. Meta-heuristic approaches tend to be computationally costly, as they must search extensively through a vast space

[1]Mobile Robotics Laboratory, Center for Intelligent Machines, School of Computer Science, McGill University, Montreal, Canada ahollid@cim.mcgill.ca

[2]Available at https://www.cim.mcgill.ca/~mrl/hgrepo/transit_learning/

of possible transit networks. But they can provide solutions of good quality on large problem instances, making them applicable in some real-world cities.

B. Deep Learning for Optimization Problems

In [3], an overview is provided of the use of deep learning for CO problems such as the Travelling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). The authors note two ways in which machine learning can be of use: in improving speed by providing fast learned approximations of operations performed repeatedly by an existing algorithm; and in directly learning novel policies for exploring the space of solutions by doing Reinforcement Learning (RL) on a cost function. Our work has elements of both. We train a model along the lines of the second category, but then employ it to replace a hand-engineered step of an existing algorithm in a way that is closer to the first category.

GNNs [11] are deep learning models designed to operate on graph-structured data. Inspired by the success of convolutional neural networks on computer vision tasks, [12] and [13] separately proposed neural graph operators, which have since been built upon by [14], [15], [16] and many others. GNNs have been applied in many domains, such as predicting chemical properties of molecules [13], [17], analyzing large web graphs [18], and in combination with RL, designing printed circuit boards [19].

Many CO problems can be readily represented as problems on graphs, making GNNs a good fit when using deep learning to solve them. [20] proposed a neural attention system they called a Pointer Network, which they trained via supervised learning to solve TSP instances. Several following works [21], [22], [23], [24], [25] built on this approach by using RL to train GNNs, and attained impressive performance on the TSP, VRP, and related problems. The quality of solutions from these deep learning methods approach those from specialized TSP algorithms such as Concorde [26], while requiring much less run-time to compute.

While neural networks have been applied to predictive tasks in urban mobility [27], [28], [29], [30], [31], [32], there have been relatively few attempts to apply deep learning or RL to transit planning. In [33], a GNN is trained to predict the equilibrium traffic assignment over road networks, and is then used as a proxy for the cost function in a genetic algorithm that designs road networks. In [34], RL is used to train a tabular dispatcher agent for a subway line to produce a dynamic schedule, and in [35] a similar approach using DRL is applied to decide when to dispatch buses on a single bus route in response to varying demand. In [36], a similar approach is taken to the more complex multi-route dispatch problem. In [37], a tabular RL method is used to learn to control passenger inflow rates on stations along a single subway line. None of these works deal with the NDP, however.

In [38], several tabular RL approaches are compared for learning how transit demand responds to iterative changes to a transit network, and an algorithm is proposed to use the resulting learned model to generate a transit network. In [39], a tabular RL approach is used to solve the NDP and FSP in tandem. While these results are intriguing, they demonstrate their approach only on the well-known Mandl benchmark [40], a very small example with only 15 nodes. Tabular RL is an approach that scales poorly with the size of problems, and is unlikely to be applicable to more realistic problem sizes.

The closest work to ours of which we are aware is [41], in which the approach of [22] is extended to plan bus routes and frequencies. But their approach requires training a separate GNN for every problem instance, and like [39], they demonstrate it only on the small Mandl benchmark. We note that we privately experimented with a similar approach, but abandoned it because it failed to learn effectively beyond this size of problem. By contrast, we show that our approach is effective in cases with as many as 127 nodes.

III. The Transit Network Design Problem

A. Problem definition

Let a **city** be represented as an augmented graph:

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

Where $\mathcal{N}$ is a set of $n = |\mathcal{N}|$ nodes where a transit route may make a stop, $\mathcal{E}_s$ is a set of weighted, un-directed edges of the form (i, j, t_{ij}) representing streets that connect these stops, and D is the $n \times n$ Origin-Destination (OD) matrix, where D_{ij} represents the demand for travel between stops i and j. The edge weight t_{ij} is the time it takes to drive along the street from node i to j or from j to i. Demand is assumed to be symmetric, with every desired trip having a corresponding return trip in the opposite direction, so D is diagonally symmetric.

A **transit route** r is a sequence of nodes making a path through $\mathcal{C}$: $r = [i, ..., j]$. A vehicle on route r visits each stop in the sequence in order, picking up and dropping off passengers at each one, and then travels back along the stops in reverse order of r (hence, all routes are assumed to be bi-directional).

In the NDP, we are given a city $\mathcal{C}$ and we seek to produce a set of transit routes $\mathcal{R}$ that minimizes some cost function $C(\mathcal{C}, \mathcal{R})$, which reflects the users's desires and priorities. Typically, this cost function depends on the total travel time over all trips and the operating cost of $\mathcal{R}$, and possibly other factors as well. An example city with proposed transit routes is shown in Fig. 1.

B. Cost functions and constraints

In this work, we impose the following constraints on the transit network $\mathcal{R}$:

1) Routes may not contain cycles: the sequence r can contain each node in $\mathcal{N}$ at most once, not counting the return trip made backwards along r.
2) Every pair of consecutive stops (i, j) on all $r \in \mathcal{R}$ must have a corresponding edge $(i, j, t_{ij}) \in \mathcal{E}_s$; stops along the route's path cannot be skipped.

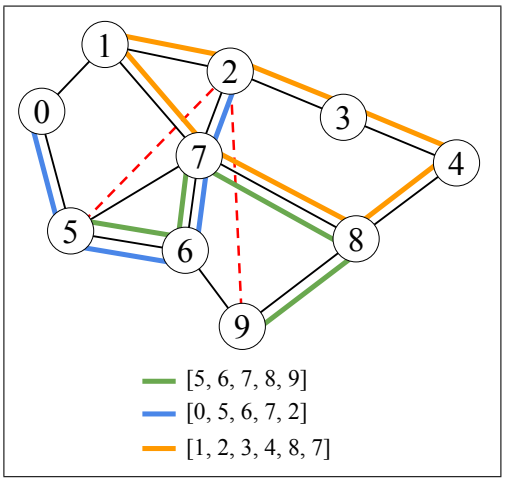


Fig. 1: An example city graph with ten numbered nodes and three routes. Street edges are black, routes are in colour, and two example demands are shown by dashed red lines. The edges of the three routes form a subgraph of the street graph $(\mathcal{N}, \mathcal{E}_s)$. All nodes are connected by this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

3) $|r| \geq MIN$ and $|r| \leq MAX$ for all $r \in \mathcal{R}$, where MIN and MAX are minimum and maximum per-route stop counts specified by the user.
4) $|\mathcal{R}| = S$, where S is a number of routes specified by the user.
5) $\mathcal{R}$ must provide some path between every pair of nodes $(i, j) \in \mathcal{N} \times \mathcal{N}$.

We denote the total time for a vehicle to traverse route r in one direction as t_r. Let A_r denote the set of street edges (i, j, t_{ij}) traversed by route r; then $t_r = \sum_{A_r} t_{ij}$.

We denote the total time of the shortest path through the transit network $\mathcal{R}$ from i to j, including time spent making transfers, as $t_{\mathcal{R}ij}$. All transfers are assumed to take a constant amount of time p_T.

We then define the cost function as a weighted sum of two components, the operator cost C_o and the passenger cost C_p. C_o is the average traversal time of all routes, and C_p is the average time of all passenger trips, assuming every passenger takes the shortest path through $\mathcal{R}$:

$$C_o(\mathcal{C}, \mathcal{R}) = \frac{\sum_{r \in \mathcal{R}} t_r}{S} \tag{2}$$

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

We also define a third term C_c which counts the number of constraint violations in $\mathcal{R}$ given city $\mathcal{C}$. The full cost function has the form given in (4).

$$C(\mathcal{C}, \mathcal{R}) = \alpha C_p(\mathcal{C}, \mathcal{R}) + \beta C_o(\mathcal{C}, \mathcal{R}) + \gamma C_c(\mathcal{C}, \mathcal{R}) \tag{4}$$

IV. Methodology

A. Algorithm

Our proposed route-planning approach is outlined in Algorithm 1. It starts with a set of node sequences called path segments, as well as information about the city $\mathcal{C}$ and any pre-existing routes $\mathcal{R}'$. The path segments used are the set of shortest paths between all node pairs SP, as this enforces a degree of directness on the proposed routes. SP can be computed efficiently via the Floyd-Warshall algorithm [42], which has time complexity $O(n^3)$. In practice we find that SP can be calculated even for $n \approx 600$ in less than one minute, and so this does not present a major computational obstacle.

The route being planned is initially empty: $r = []$. The algorithm alternates between selecting a path $p \in$ SP to add to r on odd-numbered steps $1, 3, ...$, and choosing whether or not to extend r further on even-numbered steps $2, 4, ...$ (unless $|r| < MIN$, in which case the even-numbered steps are skipped to enforce constraint 3). If at some even-numbered step the algorithm chooses to stop extending r, or no valid paths remain to extend r, r is added to $\mathcal{R}$ and the algorithm ends; otherwise, another path is chosen from SP to extend r. To plan a network of S routes, this process is simply repeated S times.

When $r = []$, any path p with $|p| \leq MAX$ may be chosen, but otherwise, considered paths are subject to the following limits. To enforce constraint 1, only paths that do not share any nodes with r are considered. To enforce constraint 2, considered paths must end at a neighbour of r's first node, or begin at a neighbor of r's last node. Finally, to enforce constraint 3, if $|p| + |r| > MAX$, p is not considered.

The choices of which path segment to add to the in-progress route at each step, and whether to halt after each addition, are made stochastically according to probabilities output by a GNN model we refer to as the **policy** (π), based on input formed from a numerical representation of $\mathcal{C}, \mathcal{R}, r, \alpha, \beta$, and the candidates p. The policy π is trained over a dataset of many cities to give high probabilities to actions that minimize $C(\mathcal{C}, \mathcal{R})$. π_{halt} refers to the component of the policy that outputs a stopping probability at even-numbered steps, and π_{ext} refers to the component that outputs probabilities for path extensions at odd-numbered steps.

Because the algorithm is stochastic, it can be run many times on a single city $\mathcal{C}$ to generate a diverse set of transit networks, and the lowest-cost of these can be selected. Alternatively, a deterministic variant of the algorithm can be used, in which the probabilities output by π are interpreted as "scores", and the option with the highest score is always chosen.

B. Graph Neural Network Training

The neural network policy π is trained on a dataset of randomly-generated cities. This is intended to cause π to learn route-generation heuristics that will be effective even in previously-unseen cities. The city and transit network

Algorithm 1 Assemble Route

Input: city graph $\mathcal{C} = (\mathcal{N}, \mathcal{E}_s, D)$, set of current routes $\mathcal{R}$, set of shortest paths SP, MIN, MAX, neural network policy π
$r \leftarrow []$
$\mathcal{A} \leftarrow \{p | p \in \text{SP}, |p| \leq MAX\}$
Halt $\leftarrow$ False
while Halt = False **and** $|\mathcal{A}| > 0$ **do**
  $\Omega \leftarrow \{\pi_{ext}(\mathcal{C}, \mathcal{R}, r, p) | p \in \mathcal{A}\}$
  $p \leftarrow$ sample $\mathcal{A}$ according to probabilities in Ω
  $r \leftarrow [p|r]$ or $[r|p]$, as appropriate
  if $|r| = MAX$ **then**
    halt $\leftarrow$ True
  else if $|r| \geq MIN$ **then**
    $P(\text{halt}) \leftarrow \pi_{halt}(\mathcal{C}, \mathcal{R}, r)$
    halt $\leftarrow$ sample {True,False} given $P(\text{halt})$
  $\mathcal{A} \leftarrow$ all paths in SP that are valid extensions to r
return r

parameters are held constant throughout training at $n = 20$, $S = 10, MIN = 2, MAX = 12$. The algorithm described in section IV-A is used to generate a transit network $\mathcal{R}$ for this city. The cost of this transit network $C(\mathcal{C}, \mathcal{R})$ is computed. In conjunction with the probabilities output by π while generating $\mathcal{R}$, the cost is used to update the internal parameters of π via the back-propagation algorithm [43], causing choices that led to lower costs to be more likely in future. Transfer time p_T was set to 300 seconds, a value commonly used in the literature ([9], [10]).

To generate the random cities used in training, we first generate $\mathcal{N}$ and $\mathcal{E}_s$ according to the same procedure used in [44] to generate the synthetic cities of the Mumford benchmark dataset. Specifically, we first uniformly sample $n = 20$ 2D points in a unit square to form $\mathcal{N}$; we then initialize $\mathcal{E}_s$ to be the minimum spanning tree of $\mathcal{N}$, and then add edges to $\mathcal{E}_s$ in order of ascending t_{ij} until $|\mathcal{E}_s| = n_e$, where n_e is a predefined number of edges. While [44] chose n_e in each instance to match that observed in some real world city, we note a very nearly linear relation between n_e and n in their examples, and so in our synthetic instances we choose $n_e = 38$, following this relation. We generate an OD matrix D with uniform random demand between every pair of nodes. Inter-node demands are sampled in the range $[60, 800]$, again following the range used in the Mumford benchmark.

The models used in our experiments are trained over a total of 147,455 cities, on a commercial desktop PC with an Intel i9-12900F processor and an NVIDIA GeForce RTX 3090 graphics processing unit. Training the model took approximately 40 minutes on this hardware.

The GNN we developed for this application is a fairly conventional deep learning model. The details are not central to the significance of this work, so we omit them here. We refer interested readers to the pre-print of our concurrent work [45], which provides a full description of the neural network's architecture, the format of its inputs, and the RL algorithm used to train it.

V. Experiments

A. Optimizers and Initialization

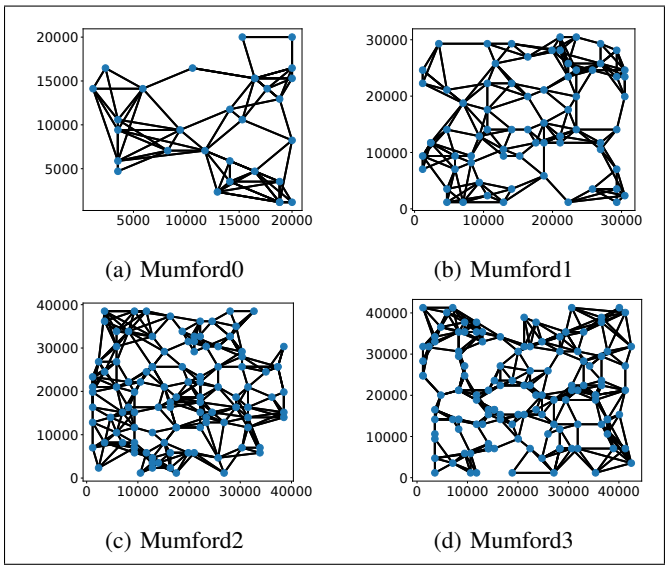
We perform experiments in which we compare our learned planner (which we denote LP) to two recent optimization algorithms: the BCO algorithm of [9], and the state-of-the-art hyper-heuristic algorithm of [10]. Both algorithms start their optimization from an initial solution, and have their own procedure for how to construct that initial solution. We compare each algorithm to a variant in which the initial solution is constructed by running our GNN policy in deterministic mode. For the full details of these algorithms, we refer the reader to the original publications.

1) Hyper-Heuristic: We use the variant of the hyper-heuristic algorithm that uses a sequence-based selection method (SS) and the Great Deluge acceptance method (GD). The authors of [10] refer to this configuration as SS-GD, and report that it has the best performance among their variants. Starting from an initial solution $\mathcal{R}_0$, the algorithm modifies the solution with a variety of heuristics for a finite number of iterations, tracking which sequences of heuristics have previously improved the solution and repeatedly applying those sequences. The initial solution $\mathcal{R}_0$ is formed in two stages: the "construction" stage, in which S routes are added one-by-one to $\mathcal{R}_0$ from SP to minimize the number of constraint violations in the transit network, and the "repair" stage, which randomly modifies $\mathcal{R}_0$, keeping each result if it does not have more violations, until no constraints are violated. We refer to this as the "plain hyper-heuristic" (PHH) approach.

In our modified version, the "construction" stage is replaced by a single run of our LP algorithm in deterministic mode: $\mathcal{R}_0 = \text{LP}(\mathcal{C}, S, MIN, MAX, \pi)$. This produces a complete transit network to which the "repair" stage is then applied, and the rest of the algorithm proceeds unmodified. We denote this the "neural initialization hyper-heuristic" (NIHH) approach.

2) Bee Colony Optimization: The Bee Colony Optimization algorithm for the NDP proposed by [9] uses B independent worker "bee" processes. Each bee starts with the same initial solution $\mathcal{R}_0$. The algorithm proceeds in two alternating phases: exploration, and recruitment. In the exploration phase, each bee randomly modifies its current solution, keeping any improvements. Then in the recruitment phase, the bees whose solutions are best are designated as recruiters. The other bees select a recruiter at random and replace their own solution with a copy of that recruiter's. These phases repeat for a user-configured number of iterations.

The initialization process for plain Bee Colony Optimization (PBCO) is similar to that for PHH, but does not involve a "repair" step, and the initial routes are chosen to maximize demand served rather than to minimize constraints violated. In the neural initialization BCO (NIBCO) variant, we replace this initialization process entirely with a single run of our LP algorithm.



(a) Mumford0 (b) Mumford1

(c) Mumford2 (d) Mumford3

Fig. 2: The road networks of the four synthetic Mumford cities. Units of all axes are in meters, and are inferred from the drive-times of edges and an assumed fixed driving speed of 15 meters per second.

B. Procedure

As is common practice in the NDP literature, we conducted two sets of experiments: one where transit networks are optimized from the passenger perspective (with $\alpha = 1, \beta = 10^{-10}$), and one where they are optimized from the operator's perspective (with $\alpha = 10^{-10}, \beta = 1$). Separate GNN policies were trained for each of these two perspectives: π_p for the passenger perspective and π_o for the operator perspective. π_o was trained with the cost weights held constant at $\alpha = 10^{-10}, \beta = 1$. By contrast, the π_p policy used in our passenger perspective experiments was trained over a range of cost weights, with α uniformly sampled from $[0, 1]$ and $\beta = 1 - \alpha$. We found that the resulting π_p achieved lower average cost on a held-out validation dataset than when it was trained only with $\alpha = 1, \beta = 10^{-10}$.

Each set of experiments is carried out on the four synthetic scenarios (cities) of the Mumford benchmark [44]. These range in size from $|\mathcal{N}| = 30$ to $|\mathcal{N}| = 127$ nodes and from $S = 12$ to $S = 60$ required routes. The same neural network policies π_p and π_o, trained on the same dataset, are applied four Mumford cities, in order to assess whether a policy trained on one dataset can be successfully applied to various unseen cities.

The numbers of nodes and routes and the values of MIN and MAX in each city are based on one real-world city [46], so these values reflect a realistic range of sizes, though we note that some cities may be much larger - to cite one example, the city of Montreal contains 8,500 bus stops covered by 212 routes [47].

The authors of [9] and [10] have not released source code for their methods, so we conduct these experiments using our own implementation of their algorithms, written in Python. Furthermore, the "SS-GD" configuration of the hyper-heuristic algorithm has two important parameters, f_0 and ΔF, but the authors do not report what values they use for these. To control for possible differences in parameter values and implementation that could affect the outcome, we compare the neural-initialization results with the results from our own implementations of PHH and PBCO, rather than comparing with the results reported in the original papers.

In our hyper-heuristic experiments, we set $f_0 = 0$, and $\Delta F = C(\mathcal{C}, \mathcal{R}_0)$. Following the original paper, we set the constraint weight $\gamma = \infty$. We run these experiments for the same number of iterations used on each city in [10]'s passenger perspective experiments: 5,000,000 on Mumford0, 3,000,000 on Mumford1, 524,000 on Mumford2, and 365,000 on Mumford3. We use these iteration counts in both our passenger-perspective and operator-perspective experiments.

In our BCO experiments, we hold all parameters of the BCO algorithm the same as in [9]'s experiments, for both the PBCO and NIBCO approaches. Since there is no "repair" step in the initialization, the initial solutions may have $C_c > 0$, so we set $\gamma = 5$ so the algorithm can follow the gradient of C towards fewer constraint violations ($\gamma = \infty$ prevents this, as any number of constraint violations then has equal cost $C = \infty$). As in [9], we run the algorithm for 400 iterations on all cities.

C. Results

We report a range of standard metrics for our experiments. The passenger metric is the average passenger trip time in minutes (ATT); The operator metric is route total time in minutes, RTT $= \sum_{\mathcal{R}} t_r$; d_k is the percentage of transit trips that required k transfers; and d_{un} is the percentage of transit trips that required 3 or more transfers.

Table I contains the results for the passenger-perspective and operator-perspective experiments. We see across both tables that LP on its own is always outperformed on the metric being optimized. However, using the policy to initialize the hyper-heuristic algorithm in NIHH improves performance on the target metric over the default initialization of PHH on all cities except the smallest, Mumford0. NIBCO improves over PBCO as well on each city, except for Mumford1 from the operator perspective.

In the passenger perspective runs, NIHH improves over PHH in ATT by 0.8% on Mumford1, 3.6% on Mumford2, and 4.2% on Mumford3. NIHH's increase over PHH in RTT in the operator perspective runs is much sharper, going from 0% on Mumford0, to 11% on Mumford1, to 17% on Mumford2 and 3. Meanwhile, NIBCO improves over BCO by 9.2% on Mumford0, 12% on Mumford1, 0.6% on Mumford2 and 10.5% on Mumford3 for the passenger perspective - the difference here is consistent across environments. For the operator perspective, the improvement on Mumford0 is minor (2.3%) and on Mumford1 NIBCO performs worse than PBCO (-12%). But for Mumford2 and Mumford3, the improvement is significant: 27% and 30% respectively.

Overall, neural initialization enhances the quality of the final solutions found by each algorithm. We note that the

TABLE I: Metrics for each method under consideration, under passenger and operator optimization perspectives. Bold values indicate which entry in a row is the best or tied for best on that metric and perspective. Entries in the d_1 and d_2 rows are not bolded as higher or lower values are not strictly better or worse, but depend on d_0 and d_{un}.

		Passenger Perspective					Operator Perspective				
City	Metric	LP	PHH	NIHH	PBCO	NIBCO	LP	PHH	NIHH	PBCO	NIBCO
Mumford0	ATT	16.60	**14.13**	**14.13**	17.39	15.92	**26.31**	32.25	31.07	32.33	29.41
	RTT	434	1576	1582	**333**	537	152	**94**	**94**	129	126
	d_0	46.70	**81.10**	80.65	41.65	53.82	**19.60**	14.34	15.96	17.37	16.68
	d_1	45.65	18.63	19.34	48.13	40.73	36.51	22.06	25.70	25.99	28.27
	d_2	7.03	0.27	0.01	10.13	5.14	28.15	26.33	23.50	14.63	28.29
	d_{un}	0.62	**0.00**	**0.00**	0.09	0.32	1.88	**0.00**	**0.00**	**0.00**	**0.00**
Mumford1	ATT	24.29	22.59	**22.42**	27.28	24.33	30.97	32.62	47.32	**29.34**	30.04
	RTT	1238	4913	5065	**886**	1438	720	489	**437**	599	683
	d_0	34.43	39.14	**41.21**	23.23	35.66	**20.34**	16.30	16.47	20.07	19.12
	d_1	46.04	58.56	57.50	42.53	42.31	31.82	32.63	23.62	43.73	31.72
	d_2	17.65	2.30	1.30	25.20	18.24	28.81	27.96	18.61	27.43	28.98
	d_{un}	1.88	**0.00**	**0.00**	9.04	3.79	19.03	23.11	41.30	**8.76**	20.18
Mumford2	ATT	26.04	25.91	**24.97**	26.08	25.91	29.94	33.81	31.70	**29.35**	36.79
	RTT	4534	18080	15375	**2818**	4215	2806	2374	1974	2472	**1947**
	d_0	39.52	32.58	**45.41**	25.62	38.60	20.59	13.21	15.11	**23.79**	11.42
	d_1	47.49	64.21	53.93	44.90	48.04	40.09	27.70	34.43	44.50	22.94
	d_2	11.90	3.21	0.66	23.49	12.77	26.87	27.91	30.14	24.95	27.45
	d_{un}	1.09	**0.00**	**0.00**	5.99	0.58	12.45	31.18	20.31	**6.77**	38.19
Mumford3	ATT	28.76	28.81	**27.59**	30.18	28.81	**31.74**	35.93	34.21	32.71	36.59
	RTT	5349	24904	19660	**4330**	5221	3628	3207	2662	3203	**2470**
	d_0	36.98	29.01	**44.37**	26.94	36.62	20.98	11.66	13.90	**21.54**	12.59
	d_1	49.49	65.62	55.13	48.50	49.68	42.11	27.59	31.09	46.32	26.61
	d_2	12.78	5.32	0.50	22.28	12.76	27.84	31.07	30.57	25.09	27.18
	d_{un}	0.74	0.05	**0.00**	2.28	0.94	9.07	29.68	24.43	**7.05**	33.62

LP took less than 10 seconds to produce the initial transit network in every instance, while the original construction stage took up to 25 seconds, so this superior initialization comes at no additional cost in run time.

Furthermore, neural initialization appears to enhance the scalability of the optimization algorithms, as it is most beneficial when $n > 100$. Many real-world cities have hundreds of possible stops, so this improvement may be very significant in terms of the applicability of these algorithms to real cities.

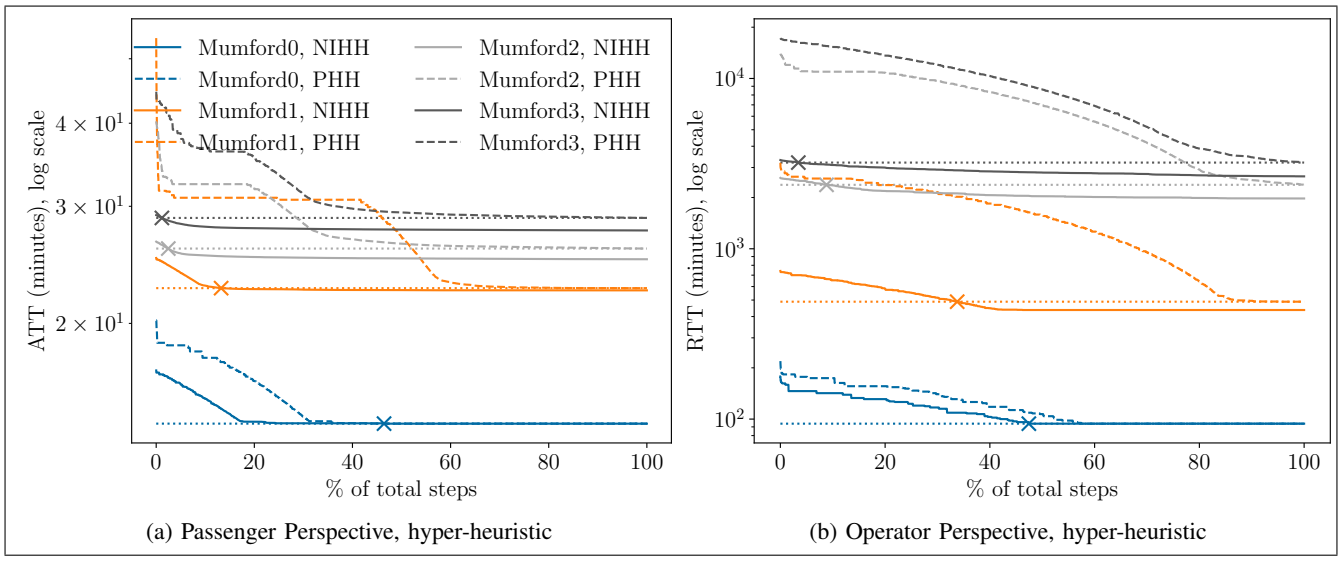
We observe also that on the two largest cities, Mumford2 and Mumford3, NIHH's final solutions are Pareto improvements on PHH: they achieve both lower passenger *and* operator cost, in each of the passenger and operator perspective cases. The values of d_0, d_1, d_2, and d_{un} show that the plans proposed by NIHH also require passengers to make fewer transfers: in both passenger and operator perspective cases on Mumford1, 2, and 3, NIHH increases d_0 and either reduces d_{un}, or reduces d_2 if $d_{un} = 0$ in PHH.

In Fig. 3, we compare the cost of the best solutions found over the course of the hyper-heuristic algorithms given the different initializations. The leftmost ends of these curves show that the initializations from the LP used in NIHH are much lower-cost than the default initializations used in PHH. We mark with an 'X' the point on each NIHH curve where it achieves the same cost as the plain algoirthm's final cost. We show this only for the hyper-heuristic variants, because in most experiments PBCO never achieved the starting cost of NIBCO at all.

As the cities get bigger, this point happens earlier in the course of NIHH's runs. In the passenger perspective case, NIHH achieves PHH's best score at 15% of the way through on Mumford1, at 4% on Mumford2, and at 2% on Mumford3; the corresponding values for the operator perspective are 35%, 10%, and 4%. This implies another use for neural initialization: improved run-time. The user can achieve the same quality of result in far less time by running optimization for fewer iterations, or can run for the same amount of time to improve on the results by a given amount, or can achieve any trade-off of improved run-time and improved cost by stopping at an intermediate number of iterations.

VI. Conclusions

While the Transit Network Design Problem is very challenging, there are regularities in its structure across different instances, enough so that widely-applicable heuristics can be learned from enough data. While these heuristics may not on their own outperform state-of-the-art meta-heuristic algorithms, they can be used to prime those algorithms, improving their results over priming them with simpler, manually-designed heuristics. Specifically, this can improve passenger trip time by up to 4.2% and operator costs by up to 17% on realistic benchmarks, without increasing the time taken to generate the solution. Alternatively, it can be used to generate solutions of the same quality as the algorithm's default initialization in as little as 2% of the running time. The user can trade off between improved run-time and solution quality simply by choosing when to halt the optimization procedure.



(a) Passenger Perspective, hyper-heuristic

(b) Operator Perspective, hyper-heuristic

Fig. 3: Optimized variable versus optimization progress for the hyper-heuristic experiments. Solid lines indicate neural initialization and dashed lines indicate default initialization, while different colours indicate different cities. Each dotted horizontal line indicates the final value achieved by default initialization, and an X marks the point, if any, where neural initialization crosses this line.

The improved solution quality offered by initializing with a neural-network generated policy is of clear practical relevance to the planning of public transit systems in real cities. Meanwhile, the major improvements in speed offered by this technique could help enable new applications, such as live and recurring refinement and re-planning of transit routes. In addition, by allow the algorithm to run much faster, additional representational enhancements may be possible that would otherwise have been too costly with traditional methods.

We have here explored one way in which deep learning planners can be used to enhance the performance of meta-heuristic methods. Other possibilities include the use of these planners as a sub-heuristic employed during meta-heuristic optimization algorithms (which we consider in our concurrent work [45]), and attempting to directly learn a policy for searching the solution space. It would also be of value to try to establish theoretical bounds on the cost achieved by this learned optimization process, compared to what is optimal. And it would be of value to adapt this technique to a multi-objective optimization context, with passenger and operator costs considered separately. We leave these possibilities to future work.

References

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

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

[3] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[4] C. L. Mumford, "Research on the urban transit routing problem (bus routing)," https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/Outline.html, 2013, accessed: 2023-03-24.

[5] C. 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.

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

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

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

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

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

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

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

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

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

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

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

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W.

Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272. [Online]. Available: https://proceedings.mlr.press/v70/gilmer17a.html

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

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

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

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

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

[23] H. Lu, X. Zhang, and S. Yang, "A learning-based iterative method for solving vehicle routing problems," in *International Conference on Learning Representations*, 2019.

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

[25] J. J. Damanik, H. Kasan, and H.-L. Choi, "Solving delivery assignment in hybrid-transit network using multi-agent reinforcement learning," in *Robot Intelligence Technology and Applications 6: Results from the 9th International Conference on Robot Intelligence Technology and Applications*. Springer, 2022, pp. 485–497.

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

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

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

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

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

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

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

[33] B. Madadi and G. H. d. A. Correia, "A hybrid deep-learning-metaheuristic framework for discrete road network design problems," *Available at SSRN 4470984*.

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

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

[36] H. Yan, Z. Cui, X. Chen, and X. Ma, "Distributed multiagent deep reinforcement learning for multiline dynamic bus timetable optimization," *IEEE Transactions on Industrial Informatics*, vol. 19, pp. 469–479, 2023.

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

[38] G. Yoon and J. Y. Chow, "A sequential transit network design algorithm with optimal learning under correlated beliefs," *arXiv preprint arXiv:2305.09452*, 2023.

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

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

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

[42] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.

[43] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

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

[45] A. Holliday and G. Dudek, "Neural bee colony optimization: A case study in public transit network design," https://www.cim.mcgill.ca/~mrl/pubs/ahollid/preprint.pdf, 5 2023.

[46] C. L. Mumford, "Download link to the mumford dataset," https://users.cs.cf.ac.uk/C.L.Mumford/Research%20Topics/UTRP/CEC2013Supp.zip, 2013, accessed: 2023-03-24.

[47] Société de transport de Montréal, "Everything about the stm," 2013, accessed: 2023-05-17. [Online]. Available: https://web.archive.org/web/20130610123159/http://www.stm.info/english/en-bref/a-toutsurlaSTM.htm