Learning to Prune: Fast Feasible Trip Generation for High-capacity Ridepooling

Youngseo Kim, Sirui Li, Hins Hu, Wenbin Ouyang, Samitha Samaranayake, Cathy Wu

Extended abstract submitted for presentation at the 12th Triennial Symposium on Transportation Analysis conference (TRISTAN XII)
June 22-27, 2025, Okinawa, Japan

February 11, 2025

Keywords: Learning-guided Discrete Optimization, High-capacity Ridepooling, Request-Trip-Vehicle Graph, Graph Neural Network

1 INTRODUCTION

The Request-Trip-Vehicle (RTV) framework is a state-of-the-art approach for solving real-time high-capacity mobility-on-demand vehicle dispatching (Alonso-Mora *et al.*, 2017). It efficiently decomposes the NP-hard problem into tractable matching and routing problems by constructing an RTV graph. The key technique for achieving tractability involves generating a set of shareable combinations of requests (referred to as a *trip*) rather than considering all combinations of requests (see fig. 1(a)). Recently, Kim *et al.* (2023) proposed an extension of the RTV framework for offline problems (i.e., problems with known demand, such as day-ahead scheduling in pre-booking systems), enabling a wider range of applications.

With increasing attention to pooling services such as microtransit (serving 4-8 passengers with shuttles and up to 30 passengers with buses) and paratransit (serving 4-12 passengers), developing efficient schedule engines for high-capacity vehicles has become more essential. However, as vehicle capacity increases, the number of trips within the framework grows exponentially, creating a bottleneck in the state-of-the-art RTV framework (Shah *et al.*, 2020). To address this challenge, we propose a supervised learning approach designed to reduce solution time by predicting and pruning infeasible trips. In this paper, we present numerical results using the Li instances (Li & Lim, 2008), a popular benchmark, which demonstrates that the proposed learning-guided RTV framework significantly reduces computation time while maintaining comparable solution quality (see fig. 1(b)). On average, 62.5% of the computation time in the RTV framework is spent on the trip generation problem filtering infeasible shared trips. Our learning-based method deploys a GNN structure to predict the feasibility of routes between trips and vehicles, achieving a median reduction in solution time of 41.9%[1], with a modest compromise of 2.6% in the service rate.

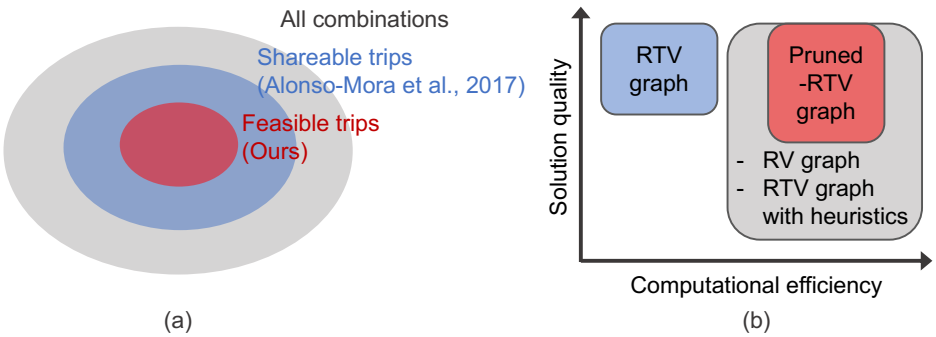

Figure 1 – *RTV vs Pruned-RTV: (a) Our method generates feasible trips as a subset of shareable trips, and (b) Pruned-RTV improves computational efficiency while maintaining solution quality.*

[1]The median for all instances is reported to mitigate the effect of outliers, as the mean can be skewed by a 3-hour timeout. For detailed results by category, please refer to Table 2 in the Results & Discussion section.

2 METHODOLOGY

2.1 Pruned-RTV framework

We consider a set of requests $\mathcal{R}$, each with specified pickup and drop-off locations and time windows, and a set of vehicles $\mathcal{V}$, each starting from a designated depot. At each decision epoch ($b = 1, \ldots, B$), we collect the *active* set of requests, whose pickup time windows begin within that decision epoch, and the *active* vehicles with available seats. Using these active sets of requests and vehicles, we generate trips $\mathcal{K}_b$ (i.e., sets of shareable requests) as proposed in the RTV framework (Alonso-Mora *et al.*, 2017). We refer to our framework as the Pruned-RTV framework to highlight our approach of predicting and pruning infeasible trips. Figure 2 illustrates a schematic overview of the Pruned-RTV graph generation.

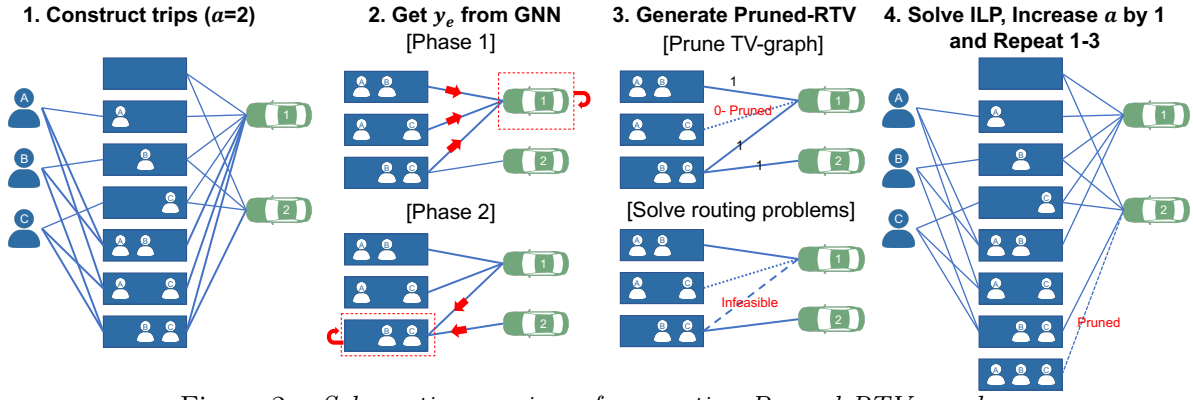

Figure 2 – *Schematic overview of generating Pruned-RTV graph*

The Pruned-RTV graph is constructed progressively, beginning with the smallest trips and incrementally expanding to larger trip sizes (i.e., trip size $a = 1, \ldots, |\mathcal{K}_b|$). This systematic process involves four main steps. First, we begin by enumerating trip sets of size a. At each batch b, all possible trips $\mathcal{K}_b$ are enumerated, excluding trips containing subsets that were previously identified to be infeasible. This iterative approach, with increasing trip size, reduces unnecessary exploration. Second, we create an edge between each trip and vehicle, where $e(k, v)$ represents each trip k and vehicle v pair, with feasibility predicted using a Graph Neural Network (GNN). A binary variable y_e indicates feasibility, with $y_e = 1$ for feasible trips and $y_e = 0$ for infeasible ones. Third, we prune all infeasible edges and then solve routing problems between trips and vehicles to determine the optimal path for a vehicle to serve all requests within a trip. This routing problem is NP-hard and often becomes a bottleneck as trip size increases, which is more likely with a large vehicle capacity c. However, the earlier pruning plays a crucial role in reducing this complexity by eliminating less promising trips, significantly reducing the number of routing problems to solve.[2] Fourth, the Integer Linear Programming (ILP) for RTV is solved to determine the optimal assignments, as proposed in Alonso-Mora *et al.* (2017). Due to the prior pruning, this step involves solving a reduced-size ILP, specifically for the Pruned-RTV graph. Afterward, we increment the trip size by 1 and repeat steps 1-4, continuing until reaching the maximum trip size $|\mathcal{K}|$ (i.e., the vehicle capacity assuming one passenger per request).

2.2 Training

2.2.1 Data

We used the Li instances (Li & Lim, 2008), a standard benchmark for the offline Pickup and Drop-off Problem with Time Window (PDPTW). Although our framework is applicable to both online (i.e., unknown, real-time demand) and offline (i.e., known demand) problems, we focus on the offline dataset to simplify evaluation. Our training process focuses on the 'lc', 'lr', 'lrc', 'LC', and 'LRC' classes, with each class name indicating a spatial distribution: 'r/R' refers to random, and 'c/C' to clustered). The dataset is split into training and validation sets in an

[2]Note that if false positives occur (i.e., if an infeasible trip is incorrectly predicted as feasible), these edges are pruned after solving the corresponding routing problem. Although this introduces a slight increase in computation time, it does not impact the overall solution quality.

8:2 ratio. We collected labeled training datasets by solving the original RTV within the rolling horizon framework (see Kim *et al.* (2023) for details) and labeling the feasibility of trip-vehicle edges. We then evaluate the out-of-distribution performance on the new 'LR' class.

2.2.2 GNN architecture

Figure 3 illustrates the architecture of a bipartite GNN used to update embeddings across multiple graph components: trip nodes (k), vehicle nodes (v), and edges (e). The model performs three sequential steps for message passing through the graph, drawing significant inspiration from the work of Morabit *et al.* (2021). In the first pass, the bipartite GNN structure identifies the neighboring trip nodes for each target vehicle node. For each trip-vehicle pair, the edge embeddings are aggregated and concatenated with the target vehicle node embedding. This joint representation is passed through the vehicle Neural Network (NN) to generate updated vehicle embeddings. The second and third passes update the trip and edge embeddings, respectively, following a similar procedure.

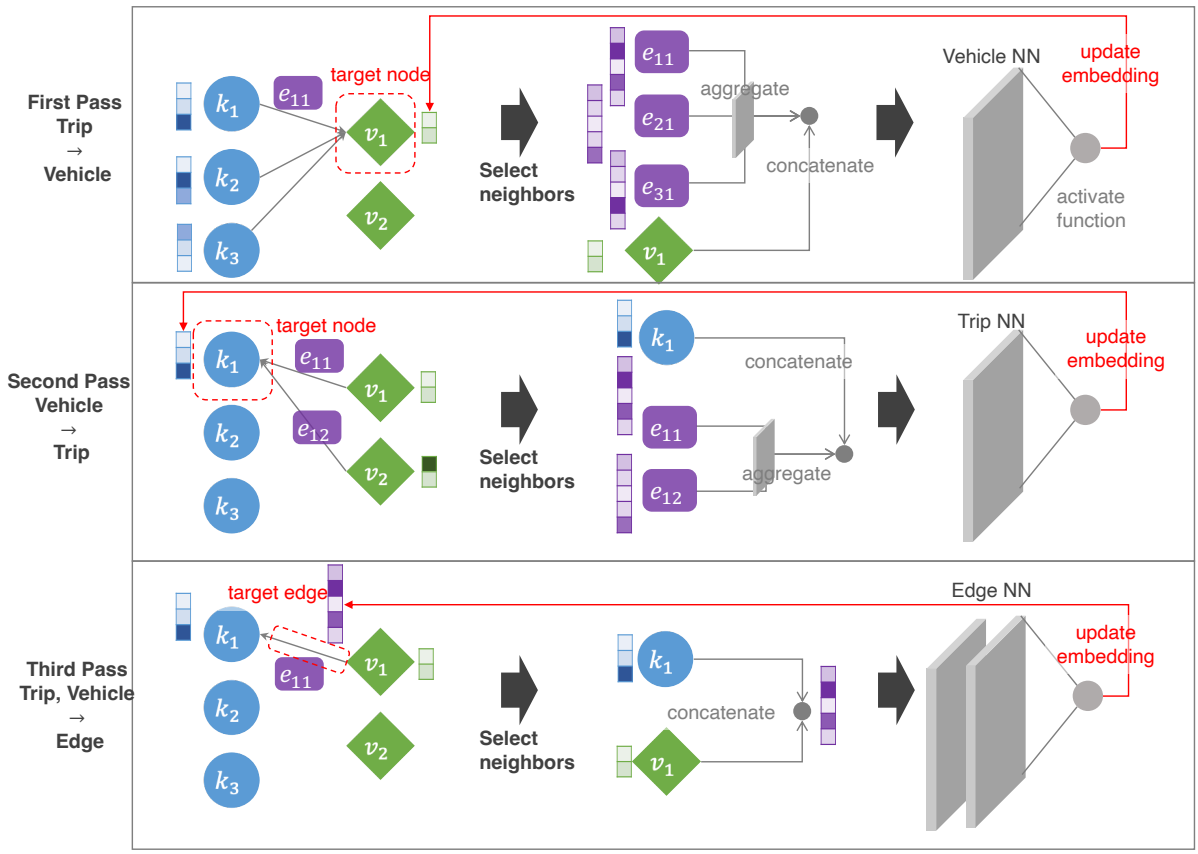

Figure 3 – *GNN architecture*

Each trip state is represented by a concatenated vector containing all request details, including the pickup and drop-off coordinates (latitude and longitude) as well as the time windows (start and end). With a vehicle capacity set to 10 passengers, this results in 80-dimensional vector (10 passengers × 8 components per request). For vehicles, feature vectors capture the position (latitude and longitude) and the corresponding time for both the current stop and the planned sequence of stops, yielding a 60-dimensional vector (3 components × 20 visited nodes). Lastly, the edge embeddings are formed by concatenating the vehicle and trip embeddings.

Table 1 – *Network parameters*

Maximum epoch	1000	Vehicle and trip NN	1 layer
Learning rate	0.0001	Edge NN	2 layers
Embedding size	32	Activation function	ReLu
Batch size	10	Weighted cross-entropy loss	10:1

All network parameters are listed in table 1. All experiment codes and data will be available online (a link will be included in the full paper). The experiments for the RTV and Pruned-RTV frameworks were conducted using Intel Xeon Platinum 8260 processors, while Intel Xeon Gold 6248 processors were used for training the GNN. Total training time is approximately 14 hours.

3 RESULTS & DISCUSSION

Table 2 compares the performance between RTV and Pruned-RTV in terms of computation time and service rate. We achieve a reduction in computation time for 31 out of 38 out-of-distribution instances of the 'LR' class, with 15 instances experiencing no performance decrease. More specifically, for the first category, which includes 15 instances, we observed both reduced computation times and improved service rates. The average computation time decreased from 8790 seconds to 1526 seconds, achieving a 5.8 times speedup. The service rate increased from 14.6% to 17.7%, a 1.2 times improvement, due to our learning-based pruning technique being more aggressive than the built-in pruning technique of RTV. For the second category, covering 16 instances, we achieved a 4.3 times speedup, with a trade-off of approximately a 20% reduction in the service rate. In future experiments, we can further minimize the reduction in service rate by lowering false negative rates (i.e., predicting feasible trips as infeasible, which deteriorates solution quality).

These results suggest promising avenues for leveraging learning techniques to improve domain-specific heuristics, such as the RTV graph, for PDPTW. For the seven instances where computation time increased, we plan to address this issue by reducing the false positive rate (i.e., predicting infeasible trips as feasible, which adds unnecessary computational load). Future research will focus on further performance improvements through training data augmentation, exploration of alternative GNN architectures, and extensive hyperparameter tuning.

Table 2 – *Comparison of performance between RTV and Pruned-RTV framework*

		RTV		Pruned-RTV		Comparison (increase by a factor of x)	
	Instance count	Compute time (sec)	Service rate (%)	Compute time (sec)	Service rate (%)	Computational speedup	Service rate increase
Computation time ↓ Service rate ↑	15	8790	14.6	1526	17.7	5.8x	1.2x
Computation time ↓ Service rate ↓	16	3663	19.6	848	15.3	4.3x	0.8x
Computation time ↑ Service rate –	4	447	12.0	5120	12.0	0.1x	1.0x
Computation time ↑ Service rate ↓	3	822	20.6	timeout*	-	-	-

* We have a 3-hour timeout to ease the repetition of experiments. This can be removed in future experiments.

References

Alonso-Mora, Javier, Samaranayake, Samitha, Wallar, Alex, Frazzoli, Emilio, & Rus, Daniela. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, **114**(3), 462–467.

Kim, Youngseo, Edirimanna, Danushka, Wilbur, Michael, Pugliese, Philip, Laszka, Aron, Dubey, Abhishek, & Samaranayake, Samitha. 2023. Rolling Horizon Based Temporal Decomposition for the Offline Pickup and Delivery Problem with Time Windows. *Proceedings of the AAAI Conference on Artificial Intelligence*, **37**(4), 5151–5159.

Li, & Lim. 2008. *Pickup and Delivery Problem with Time Windows (PDPTW) Benchmark*. https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/. Accessed: 2024-10-17.

Morabit, Mouad, Desaulniers, Guy, & Lodi, Andrea. 2021. Machine-learning–based column selection for column generation. *Transportation Science*, **55**(4), 815–831.

Shah, Sanket, Lowalekar, Meghna, & Varakantham, Pradeep. 2020. Neural approximate dynamic programming for on-demand ride-pooling. *Pages 507–515 of: Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34.