Neural Coarsening Process for Multi-level Graph Combinatorial Optimization

Hyeonah Kim[1] **Minsu Kim**[1] **Changhyun Kwon**[2] **Jinkyoo Park**[1*]
[1]Department of Industrial & Systems Engineering,
Korea Advanced Institute of Science and Technology (KAIST)
[2] Department of Industrial and Management Systems Engineering, University of South Florida
`{hyeonah_kim, min-su}@kaist.ac.kr`,
`chkwon@usf.edu`, `jinkyoo.park@kaist.ac.kr`

Abstract

Combinatorial optimization (CO) is applicable to various industrial fields, but solving CO problems is usually $\mathcal{NP}$-hard. Thus, previous studies have focused on designing heuristics to solve CO within a reasonable time. Recent advances in deep learning show the potential to automate the designing process of CO solvers by leveraging the powerful representation capability of deep neural networks. Practically, solving CO is often cast as a multi-level process; the lower-level CO problems are solved repeatedly so as to solve the upper-level CO problem. In this case, the number of iterations within the lower-level process can dramatically impact the overall process. This paper proposes a new graph learning method, *Neural Coarsening Process* (NCP), to reduce the number of graph neural network inferences for lower-level CO problems. Experimental results show that NCP effectively reduces the number of inferences as compared to fully sequential decision-making. Furthermore, NCP outperforms competitive heuristics on CVRP-CapacityCut, a subproblem of the cutting plane method for the capacitated vehicle routing problem (CVRP).

1 Introduction

Combinatorial optimization (CO) is a research area that is related to discrete mathematics, computational theory, and operations research. CO aims to find a combination of variables that minimizes the cost function (or maximize the objective function), and is applicable to various fields, including bio-synthesis [1], logistics [2], and job scheduling in the manufacturing industry [3]. Many CO problems are naturally defined on graphs, such as the maximum independent set (MIS), maximum-cut (MAX-CUT), and traveling salesman problem (TSP). We refer to such CO problems as *graph CO*. As CO problems are generally $\mathcal{NP}$-hard and intractable to solve within a limited time budget, early studies have focused on designing heuristics that generate near-optimal solutions within a reasonable time [4, 5].

Advances in deep learning relieve the reliance of CO on human-designed heuristics and problem-specific domain knowledge. The deep learning methods utilize a deep neural network (DNN) to learn the patterns of the combinatorial optimization. As deep learning automates the design process of the CO solvers via data-driven learning techniques, it overcomes the task expandability problem of the handcrafted heuristic methods. Recent studies that have used deep learning to solve graph CO problems have shown notable improvements in solution qualities and computational speeds [6, 7, 8].

[*]Corresponding author

NeurIPS 2022 New Frontiers in Graph Learning Workshop (NeurIPS GLFrontiers 2022).

Deep learning can be employed to solve CO problems by learning an end-to-end approach that maps the problem input to the solution output. End-to-end approaches have been widely explored, especially in sequential decision-making fashions [6, 9, 10, 7, 8, 11, 12, 13, 14]. These approaches leverage Recurrent Neural Network [6, 9, 10, 12], Transformer [8], and Graph Neural Network (GNN) [7, 14]. Deep learning can also be employed to solve CO problems by combining deep learning approaches with existing algorithms. For example, exact algorithms, such as the branch-and-bound and cutting plane method, can be strengthened using deep learning approaches [15, 16, 17, 18, 19, 20]. In this case, the CO problems are often tackled using a multi-level process, where the upper-level process (i.e., the main algorithm) calls the lower-level process (i.e., subroutine) to solve the main problem. Fig. 1 shows the hierarchical structure of a CO algorithm whose subroutines solve subproblems, usually CO problems, derived from the main algorithm. The main algorithm frequently calls subroutines. Thus, when one of the subroutines is replaced with deep learning-based methods, improving the subroutines can significantly impact the overall process.

We propose the *Neural Coarsening Process* (NCP), a graph learning scheme, to solve sub-CO problems. NCP employs a graph coarsening procedure that iteratively reduces the size of the graph to handle combinatorial nature. Specifically, we coarsen the graphs that has N number of nodes with the ratio γ based on the node class prediction, predict the node classes again on the coarsened graph, and then apply a simple rounding on the coarsest graph to make the final decision. The number of iterations[2] for NCP is hence $\mathcal{O}(\log N)$ as we coarsen a graph with N number of nodes.

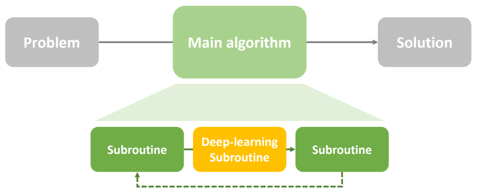


Figure 1: The main algorithm repeatedly calls the subroutines. In this case, one of the subroutines is replaced by a deep-learning method.

We evaluate the performances of NCP on *CVRP-CapacityCut*, which is the subproblem of the cutting plane method for the capacitated vehicle routing problems (CVRPs) [21]. *CVRP-CapacityCut* is a minimum-cut (MIN-CUT) problem on weighted graphs with constraints. We show that NCP performs better than fully auto-regressive method and a competitive heuristic in a large-sized *CVRP-CapacityCut* with limited number of iterations.

2 Related Works

Various works have attempted the graph CO problems by leveraging GNN with sequential decision-making [7, 22, 14, 23]. [7] proposed the Structure2Vec Deep Q-learning (S2V-DQN), a graph embedding-based reinforcement learning, which shows tremendous performance on various graph CO tasks despite requiring $\mathcal{O}(N)$ decision-making steps to terminate the Markov Decision Process (N is the number of nodes). [23] proposed a transformer-based model with an edge weight matrix, which require complex computation to process $O(N^2)$ scale of fully connected edges.

Some researchers have tried to shorten the decision steps by conducting multiple decisions simultaneously at each iteration. For example, [24] proposed learning what to defer (LwD), which decides on the actions simultaneously by leveraging locally decomposable properties. Specifically, the defer action is added to the action space and the decisions are made independently on each node. The next iteration is then conducted only with the deferred nodes. LwD shows notable performances on graph CO tasks even though it requires a task-specific transition function to satisfy the constraints.

Outside the research area of CO, the reduction of the number of DNN inferences has gained much attention, especially in text and image generation. [25] proposed the Denoising Diffusion Models in Discrete State-Spaces (D3PM), which employs a diffusion process and makes multiple decisions at each step. Similarly, [26] proposed a parallelized auto-aggressive diffusion model (Parallelized ARDM), which makes multiple decisions by learning the order-agnostic properties. These studies show remarkable results compared to the conventional auto-regressive generation by reducing the number of inferences.

[2]We refer to decoding steps within subroutines as the number of iterations unless there is a specific description.

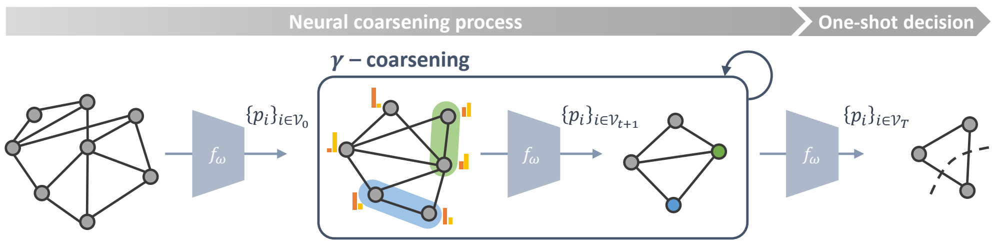

Neural coarsening process
One-shot decision
γ – coarsening
$\{p_i\}_{i\in\mathcal{V}_0}$
$\{p_i\}_{i\in\mathcal{V}_{t+1}}$
$\{p_i\}_{i\in\mathcal{V}_T}$
f_ω
f_ω
f_ω

Figure 2: The overall procedure of *Neural Coarsening Process* (NCP) in the bi-section case. NCP predicts the node class iteratively via GNN and coarsens the graph by merging the node pairs. After T iteration of γ-coarsening, a simple rule is applied to de-randomize the final prediction $f_\omega(\mathcal{G}_T)$.

3 Neural Coarsening Process

We propose the *Neural Coarsening Process* (NCP) to solve node-level graph CO tasks for subproblems with a smaller number of iterations. We employ the graph coarsening procedure to decrease the size of the input graph gradually. The graph coarsening procedure reduces a given graph $\mathcal{G}_0 = (\mathcal{V}_0, \mathcal{E}_0)$, where $\mathcal{V}_0$ and $\mathcal{E}_0$ are the nodes and edges of $\mathcal{G}_0$ respectively, and $|\mathcal{V}_0| = N$, to a sequence of smaller graphs $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_T$, such that $|\mathcal{V}_0| > |\mathcal{V}_1| > \cdots > |\mathcal{V}_T|$, where T is the number of iterations. The graph coarsening (or contraction) works by merging multiple nodes into one. Various contraction strategies can be applied according to the tasks.

The node-level graph CO tasks are node classification tasks, where the nodes are classified into one of the classes in $\mathcal{K}$. Although some tasks contain neighborhood constraints (e.g. no two adjacent nodes can share the same class in MIS), we focus on node-level tasks without the neighborhood constraints. We also apply *edge contraction* for node classification tasks[3], where a chosen edge (u, v) merges two endpoints into a single node v'. The weight of the merged nodes is the sum of the original nodes' weights, and the parallel edges are aggregated into one edge with the sum of edge values.

Overall, the node class prediction and graph coarsening procedure are conducted iteratively. The prediction model consists of a graph embedding module and a node classification module; a message passing neural network (MPNN) [27] and a Multi-layer Perceptron (MLP) are employed, respectively. The model $f_\omega : \mathcal{G} \to \mathbb{R}^{|\mathcal{K}|\times|\mathcal{V}|}$ gives an independent probability p_i^k that decides if node i will be included in class $k \in \mathcal{K}$. Specifically, we repeat following steps:

Step1. For each node i, predict the node class probability p_i^k via $f_\omega(\mathcal{G}_t)$.

Step2. Coarsens $\mathcal{G}_t$ to get $\mathcal{G}_{t+1}$.

Step2-1. Greedily select an edge according to probability q_{ij} that decides if its endpoints are in the same class, i.e.,

$$\arg\max_{(i,j)\in\mathcal{E}_t} q_{ij}, \quad \text{where } q_{ij} = \sum_{k\in\mathcal{K}} p_i^k p_j^k.$$

Step2-2. Contract the selected edge and repeat **Step2** until $|\mathcal{V}_{t+1}| = \lfloor\gamma|\mathcal{V}_t|\rfloor$.

Step1 corresponds to the node class prediction, and **Step2** corresponds to the graph coarsening with edge contractions. Note that **Step2** can be processed in parallel. The overall procedure is illustrated in Fig. 2. Since the graph is coarsened with the ratio γ for every GNN inference, the number of GNN inferences is bounded to $\mathcal{O}(\log|\mathcal{V}|)$. As the coarsened graph has fewer nodes and contains aggregated information, the node classification becomes less complicated. When the graph has been coarsened enough, we apply a rounding scheme to de-randomize the final prediction and map the nodes of the coarsened graph $\mathcal{G}_T$ to that of the original graph $\mathcal{G}_0$.

We train the model by minimizing the difference between the predicted node class probability and the true labels. Binary labels $\boldsymbol{y} = \{y_i^k\}_{i\in\mathcal{V},k\in\mathcal{K}}$ are generated in advance using the exact (or good enough) algorithms with relatively small-sized problems. For each coarsening step, a collection

[3] *Star contraction* selects a node and merge it with its neighboring nodes. *Star contraction* is applicable to MIS and we leave this in the future work

of pairs $\mathcal{D} = \{\mathcal{G}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=0}^{N}$ are gathered, and the parameters are updated to minimize binary cross-entropy (BCE) loss:

$$\mathcal{L}(\omega) = \sum_{(\mathcal{G},\boldsymbol{y})\in\mathcal{D}} \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{V}} \left[y_i^k \cdot \log p_i^k + (1-y_i^k)\cdot \log(1-p_i^k)\right]. \tag{1}$$

When the model is trained, the labels can become invalid when the graphs are coarsened due to inaccurate prediction p_i^k (e.g. the nodes with different labels are merged into one). To keep the labels valid in the entire coarsening process, we compute q_{ij} based on y_i instead of p_i. Consequently, the contraction probabilities of crossing edges, whose endpoints have different labels, are guaranteed to be all zero at the training phase.

4 Experimental Results

4.1 Target Task: *CVRP-CapacityCut*

In this study, we conduct experiments focused on the cutting plane method for CVRP. CVRP is represented as a tuple (G, K, Q), where $G = (V, E)$ is a complete graph whose node set V corresponds to the union of the depot and customer nodes, and K is the number of vehicles with capacity Q. CVRP aims to find the routes with the minimum cost for K vehicles where: (i) each customer must be served by only one vehicle; (ii) no vehicle can serve a set of customers whose total demand exceeds the vehicle's capacity Q; and (iii) every route starts and ends at the depot.

The cutting plane method is generally used to handle the exponentially-large capacity constraints. Fig. 3 shows the overall procedure of the cutting plane method for capacity constraints. First, it relaxes the problem by dropping all capacity and integer constraints. Second, it iteratively (a) solves the relaxed linear problem, (b) finds the capacity constraints violated by the current linear solution, and (c) appends them to the relaxed problem. However, designing an effective algorithm for the subroutine (b) is challenging; thus, we replace this subroutine with NCP.

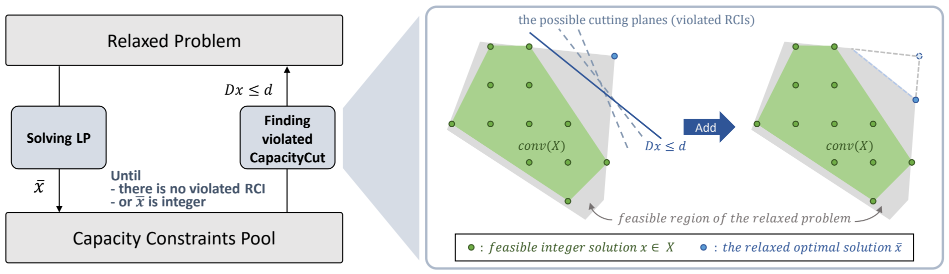

Relaxed Problem
$Dx \le d$
Solving LP
Finding violated CapacityCut
$\bar{x}$
Until
- there is no violated RCI
- or $\bar{x}$ is integer
Capacity Constraints Pool
the possible cutting planes (violated RCIs)
conv(X)
$Dx \le d$
Add
conv(X)
feasible region of the relaxed problem
● : feasible integer solution $x \in X$
● : the relaxed optimal solution $\bar{x}$

Figure 3: The overall procedure of the cutting plane method: the relaxed problem does not contain any capacity constraints at the beginning. *CVRP-CapacityCut* aims to find the capacity constraints violated by the current relaxed solution $\bar{x}_{ij}$, and add them to the relaxed problem. Adding capacity constraints makes the feasible region of the relaxed problems tighter.

CVRP-CapacityCut is defined as finding a subset $S \subseteq V \setminus \{0\}$ that violates the corresponding capacity constraints; it is highly related to the MIN-CUT problem on a weighted graph. See Appendix A for mathematical formulations and further details. *CVRP-CapacityCut* is a graph CO problem and solving this is known as $\mathcal{NP}$-hard [28].

4.2 Setup

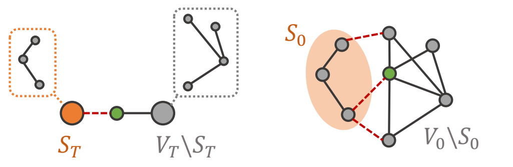
We apply NCP to solve the *CVRP-CapacityCut* efficiently. To be specific, we construct a sparse graph $G_0 = (V_0, E_0)$, where $V_0 = V$ and $E_0 = \{(i,j) \in E : \bar{x}_{ij} > 0\}$ based on the given relaxed solution $\bar{x}_{ij}$. *CVRP-CapacityCut* is a node classification task that has two classes (i.e., S and $V \setminus S$), which minimizes the total weight of the crossing edges connecting two nodes of different sets (i.e., one of the endpoints belongs to S and the other belongs to $V \setminus S$).

We let $p_i = 1$ if node i is included in S, and $p_i = 0$ otherwise to simplify the notations. The depot node, indexed as 0, is excluded from S by the definition of *CVRP-CapacityCut* (i.e. $p_0 = 0$). G_0 is coarsened according to the procedure defined in Section 3 until there are three nodes remaining (the depot and two aggregated nodes). We set q_{ij} for the depot connected edges not to coarsen the depot as follows:

$$q_{ij} = \begin{cases} p_i p_j + (1-p_i)(1-p_j) & \text{if } i, j \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Thus, the coarsening also terminates when there are no edges to contract.

After T steps of coarsening, we compute the final node selection probability $p_i = f_\omega(G_T)$, then simply round the probability to map the probabilities to the binary values. If there are no nodes with $p_i > 0.5$, we set the node that has the highest probability as 1. By doing this, the nodes of G_T are classified as the nodes in set S_T and in its complement set $V_T \setminus S_T$. As we keep tracking the node information in the coarsening process, we can map the class of the coarsened nodes to the class of the original nodes directly.



(a) The coarsest graph G_T. (b) The original graph G_0.

Figure 4: The nodes of G_T are classified into two sets S_T and $V_T \setminus S_T$, then directly mapped to G_0. The green node indicates the depot.

The training data is gathered by conducting the cutting plane method with the exact separation algorithm [21] for relatively small-size, randomly generated CVRP ($|V| \in [50, 100]$). As the collected labels are highly imbalanced, a positive weighted BCE loss is employed. The positive weight is defined as the ratio between the number of negative and positive labels.

4.3 Performance Evaluation

Baseline. We use the CVRPSEP library [29] written in C++ for the baseline separation algorithm. It consists of four heuristics, one simple *connected component* algorithm, and three *shrinking*-based heuristics. We set the maximum number of cuts for a given relaxed solution as the number of vehicles K, the same as the proposed algorithm.

Lower bound (LB) gap. Since the scale of cost varies depending on problem instances, we measure the relative performances by calculating the lower bound gap, GAP, as follows:

$$GAP = \frac{(OPT - LB)}{OPT} \times 100(\%), \tag{3}$$

where OPT is the known optimal cost and LB is the resulting cost with the cutting plane method. OPT is challenging to compute and unknown in large-scale CVRP. Hence, the solution calculated via the hybrid genetic search algorithm (HGS [4]) is used instead of OPT.

Evaluation. The cutting plane method terminates when the algorithm cannot find any violated capacity constraints, or the number of subroutine calls reaches the limit. The relaxed solution and its cost (i.e. resulting lower bound) are computed with the constraints appended at the end. CVRPSEP has at least $\mathcal{O}(2^{|V|})$ time complexity in the worst case since the connected component heuristic needs to check all the components, their complements, and unions (refer [29] for details). Thus, the evaluation with limited calls does not give any advantages to our method.

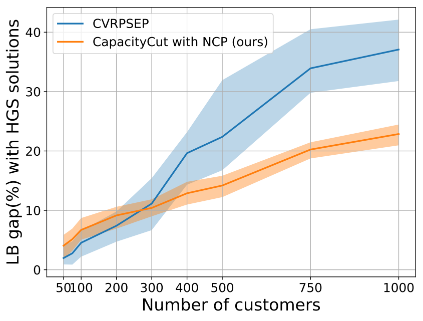
4.4 Results

Since the exact labels for the large-scale instances are intractable, the model is trained with labels generated from relatively small-sized CVRP. We evaluate the lower bound for CVRP with different sizes using both CVRPSEP and NCP to verify the performance and scalability of our model. We conduct the experiments for 10 CVRP instances of each size with limited subroutine calls.

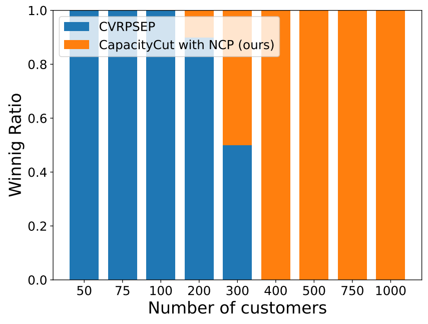
Table 1 shows the overall results of the cutting plane method with CVRPSEP and NCP. As shown in Table 1, the cutting plane method with NCP performs more effectively in larger instances, even though the model is trained in the $[50, 100]$ range. The average improvement of LB per *CVRP-CapacityCut* call (denoted as Avg. Δ LB) is calculated to evaluate the quality of added constraints. This is done by dividing the total LB improvements by the number of calls. Table 1 shows that the LB improvements are more significant with the constraints found by NCP as compared to that of CVRPSEP. We provide additional results with limited wall clock time in Appendix C.1.

Table 1: The mean and standard deviation of lower bound with iteration subroutine calls.

Method	Size	Mean LB	Std. LB	Avg. # of Calls	Avg. Δ LB
CVRPSEP	50	**9,363.594**	2,490.687	68.3 ($\leq$ 200)	38.381
	75	**13,355.482**	7,288.836	115.7 ($\leq$ 200)	34.930
	100	**15,935.360**	5,152.466	160.1 ($\leq$ 200)	34.676
	200	**21,085.211**	4,527.548	200 ($\leq$ 200)	39.401
	300	30,481.408	11,276.405	200 ($\leq$ 200)	62.129
	400	40,017.198	11,751.322	100 ($\leq$ 100)	165.266
	500	47,978.849	22,786.976	100 ($\leq$ 100)	177.991
	750	60,950.387	19,835.406	50 ($\leq$ 50)	472.611
	1,000	59,623.222	12,326.330	50 ($\leq$ 50)	432.736
CapacityCut with NCP (ours)	50	9,129.823	2,474.901	37.8 ($\leq$ 200)	61.473
	75	13,064.861	7,281.095	60.8 ($\leq$ 200)	61.263
	100	15,593.334	5,170.822	94 ($\leq$ 200)	56.308
	200	20,742.544	4,728.970	126.7 ($\leq$ 200)	58.559
	300	**31,092.012**	12,510.644	157.7 ($\leq$ 200)	75.812
	400	**43,896.118**	14,509.259	100 ($\leq$ 100)	204.055
	500	**53,865.885**	27,457.718	100 ($\leq$ 100)	234.865
	750	**73,652.108**	23,963.772	50 ($\leq$ 50)	726.645
	1,000	**73,140.790**	15,059.967	50 ($\leq$ 50)	703.088



(a) The range of LB gap for each test size.



(b) The winning ratio out of 10 instances.

Figure 5: The results of the cutting plane method with NCP and CVRPSEP.

The lower bound gap of NCP and CVRPSEP are plotted as shown in Fig. 5a. In addition, the number of times that the algorithms defeat each other out of the 10 instances (i.e., winning ratio) is measured for each size as shown in Fig. 5b. The results indicate that our algorithm performs better in large-sized CVRP ($\geq$ 300) when averaged. Furthermore, NCP outperforms CVRPSEP for every instances with more than 300 nodes. More experiments that verify the transferability of NCP are conducted using CVRPLIB X-instances [30] generated using different distributions. The results show that NCP performs well for out-distribution problems (details are in Appendix C.2).

Fig. 6 illustrates the LB gap of NCP and CVRPSEP according to the number of subroutine calls. The LB gap is significantly reduced in the early stages, and the improvement diminishes as the capacity cuts are added. Fig. 6 also shows that NCP converges faster than CVRPSEP. When the number of nodes is small, CVRPSEP converges to a lower point than NCP eventually. However, when the number of nodes is large, NCP decreases to a lower point than CVRPSEP within the limited subroutine calls. Since NCP converges faster, it can be more efficient in improving the lower bound, especially in the large-scale CVRP.

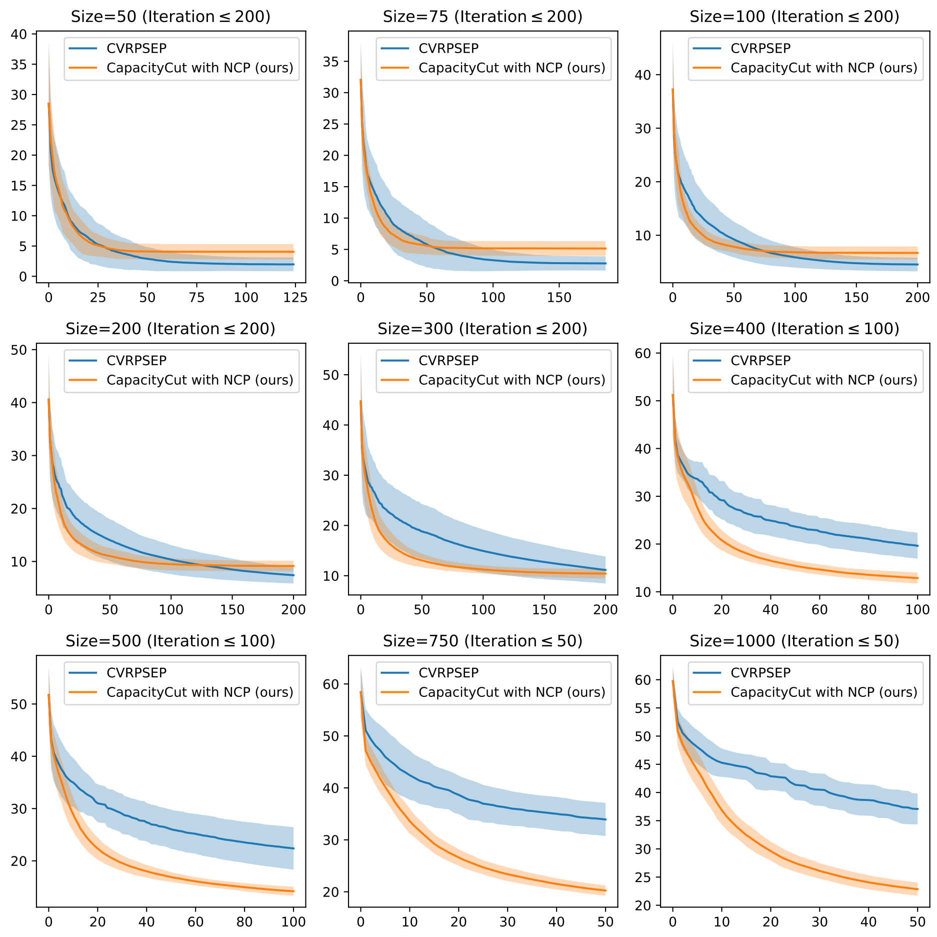


Figure 6: The LB gap according to the number of *CVRP-CapacityCut* calls. The solid line represents the average value of 10 instances and the shade areas represents the range of the LB gap.

4.5 Results for the subproblem

We compare NCP with a fully auto-regressive and an one-shot prediction model to verify the effect of NCP on *CVRP-CapacityCut*. A fully auto-regressive model makes decisions node by node based on the previous decisions; thus, decisions for the next node are conditioned by the nodes that are decided prior to it. On the other hand, an one-shot prediction model predicts probabilities of being included in S for all nodes simultaneously. One-shot prediction equals to NCP with $\gamma = 1$ since we apply the same rounding rule to get binary value. We use the same network architecture (i.e. MPNN + MLP) and train to minimize the differences using the labels (see Appendix B.4 and Appendix B.5 for details).

We evaluate the models using 100 test data generated in advance for each size of $[50, 75, 100, 200]$. The number of inferences (Avg. Steps), the cost gap of the subproblem with labels (Cost Gap), and the ratio of feasible solutions for *CVRP-CapacityCut* (Feasibility) are measured for each size. Table 2 results clearly show that our method requires a fewer number of inferences for decisions with less satisfaction for the constraints.

Fig. 7 illustrates the lower bound gap computed with the cutting plane method via different algorithms for randomly generated CVRP in the $[50, 500]$ range. We follow the evaluation process described in Section 4.3. The results show that NCP achieves a lower cost gap in subproblems even though it fails to satisfy constraints occasionally, which leads to better performances in the main CVRP. The fully auto-regressive decoding strictly satisfies the constraints of the *CVRP-CapacityCut* as the unavailable choices are masked when the decisions are made sequentially. It is challenging to

Table 2: The results for *CVRP-CapacityCut* with different decoding strategies. The subproblem's cost gap is computed based on the cost with labels.

Size	50			75		
	Avg. Steps	Cost Gap	Feasibility	Avg. Steps	Cost Gap	Feasibility
Auto-regressive	25.148	0.305	**1.0**	37.494	0.276	**1.0**
One-shot	1.0	0.256	0.687	1.0	0.657	0.758
Coarsening (ours)	16.000	**0.094**	0.635	16.460	**0.221**	0.755

Size	100			200		
	Avg. Steps	Cost Gap	Feasibility	Avg. Steps	Cost Gap	Feasibility
Auto-regressive	50.691	0.351	**1.0**	99.998	0.466	**1.0**
One-shot	1.0	0.669	0.645	1.0	4.425	0.691
Coarsening (ours)	17.700	**0.190**	0.610	19.040	**0.237**	0.575

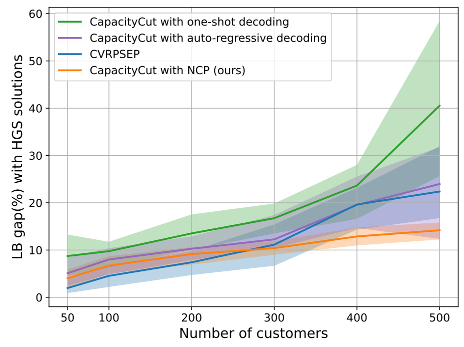


Figure 7: The range of lower bound gap of CVRP with different strategies.

consider the relationship between the nodes (i.e. variables) in the one-shot prediction, hence its performance deteriorates as it classifies the nodes jointly.

5 Conclusion

In this work, we proposed the *Neural Coarsening Process* (NCP), an effective graph learning method for subproblem of the hierarchical structures in graph CO. Our main contribution is that our method (i) requires a smaller number of inference iterations and (ii) gives a more powerful performances, than the fully auto-regressive method. The reduced iteration (i.e. the number of GNN inferences) of subproblems can have great effects on the overall procedure since the subproblems are called repeatedly. The extensive experiments show that our method outperforms both the graph learning-based auto-regressive method and competitive heuristic method on *CVRP-CapacityCut*.

Acknowledgments and Disclosure of Funding

This research was partially supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (Project No. 2022-0-01032) and the National Research Foundation of Korea (Project No. 2021H1D3A2A01039401).

References

[1] Gita Naseri and Mattheos AG Koffas. Application of combinatorial optimization strategies in synthetic biology. *Nature communications*, 11(1):1–14, 2020.

[2] Abdelkader Sbihi and Richard W Eglese. Combinatorial optimization and green logistics. *Annals of Operations Research*, 175(1):159–175, 2010.

[3] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & industrial engineering*, 30(4):983–997, 1996.

[4] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[5] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12 2017.

[6] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700. Curran Associates, Inc., 2015.

[7] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 6348–6358. Curran Associates, Inc., 2017.

[8] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

[9] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017.

[10] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[11] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

[12] André Hottung, Bhanu Bhandari, and Kevin Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2020.

[13] Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, 2021.

[14] Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

[15] Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.

[16] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2018.

[17] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[18] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33:18087–18097, 2020.

[19] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

[20] Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

[21] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

[22] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[23] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34:5138–5149, 2021.

[24] Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 134–144. PMLR, 13–18 Jul 2020.

[25] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[26] Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2021.

[27] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[28] Ibrahima Diarrassouba. On the complexity of the separation problem for rounded capacity inequalities. *Discrete Optimization*, 25:86–104, 2017.

[29] Jens Lysgaard, Adam N Letchford, and Richard W Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, 2004.

[30] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

A Explanation of *CVRP-CapacityCut*

A.1 Mathematical formulation of CVRP

Mathematically, CVRP is written in integer programming (IP). Following [29], we let x_{ij} represent the number of travels between nodes i and j. Given a subset of customers $S \subseteq V_C$, $\delta(S)$ denotes a set of *crossing* edges, one node of which belongs to S and the other belongs to $V \setminus S$. For arbitrary given edge subset $F \subseteq E$, $x(F)$ denotes the sum of travels of F, i.e. $\sum_{(i,j)\in F} x_{ij}$. The *two-index formulation* of CVRP is:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} && && (4)\\
\text{Subject to} \quad & x(\delta(\{i\})) = 2 && \forall i \in V_C && (5)\\
& x(\delta(S)) \geq 2BP(S) && \forall S \subseteq V_C && (6)\\
& x_{ij} \in \{0, 1\} && \forall 1 \leq i < j \leq |V| && (7)\\
& x_{0j} \in \{0, 1, 2\} && \forall j \in V_C, && (8)
\end{aligned}
$$

where $r(S)$ is the minimum number of required vehicles to serve the customers in S. The equality in Eq. (5) impose that the total traveling time of each node is exactly 2, so each customer is visited once by only one vehicle. The *capacity constraints* in Eq. (6) ensure that the summation of demand cannot exceed the capacity Q and a route have the depot as the connected nodes (i.e., the sub-tour elimination). The right-hand side $r(S)$ is often computed with $\lceil\sum_{i\in S} d_i/Q\rceil$, called *rounded capacity inequalities* (RCIs). The two last constraints in Eqs. (7) and (8) are the integrality conditions of the decision variables respectively. Note that if the vehicle visits only one customer, the edge between the depot and the customer j has to be traveled two times (i.e., $x_{0j} = 2$).

A.2 Cutting plane method for CVRP

In general, the *cutting plane* method is an iterative algorithm that first solves a relaxed problem, which is the problem where the hard-to-handle constraints are ignored, and then, out of the ignored constraints, the violated constraints (i.e., the cuts or cutting planes) are appended to the relaxed problem to form the new relaxed problem. The iteration terminates when the solution of the relaxed problem (i.e. the relaxed solution) becomes feasible to the original problem. Specifically in CVRP, the cutting plane method often relaxes the RCIs and the integer conditions (e.g. [29]).

In the two-index formulation of CVRP, the relaxed problem for the cutting plane method is as follows:

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} && && (9)\\
\text{Subject to} \quad & x(\delta(\{i\})) = 2 && \forall i \in V_C && (10)\\
& 0 \leq x_{ij} \leq 1 && \forall 1 \leq i < j \leq |V| && (11)\\
& 0 \leq x_{0j} \leq 2 && \forall j \in V_C && (12)\\
& x_{ij} \in \mathbb{R} && \forall i, j \in V && (13)
\end{aligned}
$$

Note that, in the relaxed problem, RCIs in Eq. (6) and the integer constraints in Eqs. (7) and (8) are eliminated, and it becomes linear programming (LP), whose optimal solution can be relatively easily found than the original problem.

In practice, to keep the relaxed problem to be LP, the cutting planes are selected among RCIs. The relaxed problem does not enforce the RCIs, so the (fractional) solution (i.e., $\bar{x}$ in Fig. 3) for the relaxed problem are not feasible for the original IP problem Eq. (4)–Eq. (8). To refine the feasible region of the relaxed problem, the *separation algorithms* find a RCI constraint which are violated by the current solution (i.e., $D\bar{x} > d$). The found inequality $Dx \leq d$ is one of the relaxed RCIs, so the original feasible solutions satisfy the inequality. Thus, this constraint *separates* the current solution from the feasible solutions of the original problems. Once the separating RCI(s) are found, the RCI(s) are appended to the relaxed problem.

A.3 The exact RCI separation

At each separation step, there exist a lot of valid inequalities (i.e. the possible cutting planes) that separate the relaxed optimal solution from the set of the integer feasible solutions. The *exact separation algorithm* [21] is designed to produce the tightest cutting planes (i.e., the cut closest to convex hull of the feasible region). The exact separation can deeply cut the relaxed feasible region by adding the tightest cutting plane into the current relaxed problem, whereas most heuristic separations are designed to find one of the possible cutting planes. By iteratively cutting the current relaxed solution out, we can get the more refined feasible region of the relaxed problem.

The exact separation is formulated as a mixed-integer programming (MIP). Given the current relaxed solution $\bar{x}$, we define $\bar{G} = (V, \bar{E})$ where $\bar{E} = \{(i,j) \in E : \bar{x}_{ij} > 0\}$ as the support graph, y_i as the binary variable whether node i is in S, and w_{ij} as the continuous variable that equals to 1 if $(i,j) \in \delta(S)$. For each $M \in \{0, \dots, \lceil \sum_{i \in V_C} d_i / Q \rceil - 1\}$, we can get the minimum crossing edge cost $z(M)$ by solving the following:

$$
\begin{aligned}
z(M) = \text{Minimize} \quad & \sum_{(i,j) \in \bar{E}} \bar{x}_{ij} w_{ij} && && (14)\\
\text{Subject to} \quad & w_{ij} \geq y_i - y_j && \forall (i,j) \in \bar{E} && (15)\\
& w_{ij} \geq y_i - y_j && \forall (i,j) \in \bar{E} && (16)\\
& \sum_{i \in V_C} d_i y_i \geq (M \cdot Q) + 1 && && (17)\\
& y_0 = 0 && && (18)\\
& y_i \in \{0, 1\} && \forall i \in V_C && (19)\\
& w_{ij} \geq 0 && \forall (i,j) \in \bar{E} && (20)
\end{aligned}
$$

The logical constraints Eq. (15) and Eq. (16) means w_{ij} has to be 1 if the y_i and y_j have different values, which is the definition of w_{ij}. For the given M, the inequality Eq. (17) enforces that the total demand of the selected nodes should exceed the entire load of M vehicles. In other words, the subset S requires at least $M + 1$ vehicles to satisfy all demands in S (i.e. the subset S requires $2(M + 1)$ times of traveling). As the minimum crossing edge weight $z(M)$ corresponds to the total traveling number of $S^* = \{i \in V_C : y_i^* = 1\}$, we can find the corresponding capacity inequality $z = \bar{x}(\delta(S^*)) \geq \sum_{i \in S^*} d_i$. Thus, if $z(M)$ is less than $2(M+1)$, the corresponding RCI is violated.

B Detailed Experimental Setting

B.1 Graph representation

Since the message passing operation of MPNN is conducted along the edges, the messages from the isolated nodes cannot be exchanged. In order to make the support graph connected while keeping the sparsity, edges between the depot and customers are added (i.e. the depot is completely connected to all customers). The modified graph is defined as $\bar{G}' = (V, \bar{E}')$, where $\bar{E}' = \{(i,j) \in E : \bar{x}_{ij} > 0 \text{ or } i = 0\}$.

Now, we define the node and edge features of $\bar{G}'$ to represent the given RCI separation problem Eq. (14)-Eq. (20). The problem has four parameters, cost coefficient $\bar{x}$, constraint coefficient d, and the right-hand side (RHS) M and Q. These parameters are not included in $\bar{G}'$, so we formulate these as the node and edge features.

Node feature The constraint Eq. (17) is related to the node decision variable y. It can be reformulated as $\sum_{i \in V_C} \frac{d_i}{Q} y_i > M$, and the coefficient $\frac{d_i}{Q}$ and RHS M define the node feature. Note that $\frac{d_i}{Q}$ has the range $[0, 1]$ by definition, but the range of M varies on the number of vehicles. Thus, we normalize it by dividing the vehicle number K (i.e., $\frac{M}{K} \in [0, 1]$). The node feature of $i \in V$ is defined as:

$$s_i = \left(\frac{d_i}{Q}, \frac{M}{K} \right) \tag{21}$$

Edge feature The cost function Eq. (14) are related to the edge decision variable w_{ij}. We directly use the cost coefficient $\bar{x}$ as an edge feature. Note that $\bar{x}_{ij}$ is 0 for the additional edges $(i,j) \in \bar{E}' \setminus \bar{E}$, and every $\bar{x}_{ij}$ are in the range $[0,1]$ by definition. The edge feature of $(i,j) \in \bar{E}'$ is defined as:

$$s_{ij} = (\bar{x}_{ij}) \tag{22}$$

B.2 Network architecture

We employ a message passing neural network (MPNN) and a simple MLP for representation learning for the graph embedding and node classification modules. Single linear layers with 128 latent dimension are employed for the node and edge initial embedding. The MPNN consists of 5 graph layers whose node update and message generation functions are MLP with $[64, 32]$ hidden dimensions. The policy module is a simple MLP with $[64, 32]$ hidden dimension. Note that it has a 1 dimension for the output and employs a sigmoid activation function since the output corresponds to the probability.

B.3 Data generation

To get exact labels for separation problems, we generate random CVRP instances following [30] and apply the cutting plane method with the exact separation algorithm described in Appendix A.3. The random CVRP instances are generated with uniformly distributed depot and customer positions, and each customer's demand is sampled from a uniform distribution range of $[1, 100]$. Since the computation of the exact RCI separation problems with a large number of CVRP is intractable ($\mathcal{NP}$-complete), we only solve the relatively small size of CVRP instances ($|V| \in [50, 100]$).

Support graphs and exact labels for the separation problems are gathered in an offline manner. Given LP solution $\bar{x}$, for each $M = 0, \ldots, \lceil \sum_{i \in V_C} d_i / Q \rceil - 1$, the exact separation algorithm solves MIP Eqs. (14) to (20). Since binary variable y_i indicates whether the node i is in S, we can set the exact node label $\hat{y}_i$ as the optimal values of y_i^*. We include label $\hat{\boldsymbol{y}} = [\hat{y}_0, \ldots, \hat{y}_{|V|}]$ into training data even if the minimum crossing edge set does not violate RCI. Hence, the policy learns how to find the minimum crossing edge set by amortizing the computation of the MIP solver. To get the next separation problem, the violated RCIs are added to the current relaxed problem, and we repeat this until the algorithm fails to find violated RCIs.

To evaluate the trained model, we generate CVRP test instances with different numbers of customers from 50 to $1,000$ (10 instances for each size). We use the identical distributions with training for depot position, customer position, and demand with CVRP training instances.

B.4 Auto-regressive decoding model

Model. First, we add a dummy node that indicates the `<end of selection>` and is connected all nodes so as to gather information about the current set composition (i.e. the dummy node has incoming edges only). We set the weights of dummy-connected edges as 0, and add an edge type feature that indicates that its source node is currently included in S. The dummy nodes are masked to available only when the selected nodes are satisfy constraints at each step.

Training. We train the model using the same labeled data with NCP. At each step, the model is trained to imitate the probability $\hat{p}_i$, calculated via the exact labels as follows:

$$\hat{p}_i = \begin{cases} \frac{\hat{y}_i}{\sum_{i \in V_c \setminus S} \hat{y}_i} & \text{if } i \neq dummy \\ 1 - \sum_{j \in |V_C|} \hat{p}_j & \text{otherwise} \end{cases}$$

We employ the mean squared error (MSE) loss because it gives better performances than the cross entropy loss.

B.5 One-shot decoding model

Model. We construct the same input graph with NCP and predict the independent node selection probabilities directly. Since the model computes the node selection probabilities independently, we can use the exact label $\hat{y}_i$ as true probability for node i. To de-randomize the resulting graph, we employ the same rounding scheme with NCP. The one-shot decoding model is identical to the graph coarsening network with the coarsening ratio $\gamma = 1$.

Algorithm 1: CapacityCut with the auto-regressive decoding

```
Input: Augmented graph $G$
Output: Selected nodes set $S$
1  Initialize set $S = \emptyset$;
2  for $step \leftarrow 0$ to $|V_C| - 1$ do
3      Predict node selection probability $\{p_i\}_{i \in V_C \setminus S} \leftarrow f_\omega(G, S)$;
4      if training then
5          Update $\theta \leftarrow \nabla \mathcal{L}(\{p_i\}_{i \in V_C \setminus S}, \{\hat{p}_i\}_{i \in V_C \setminus S})$;
6      Randomly select a node $j$ according to $\{\hat{p}_i\}_{i \in V_C \setminus S}$;
7      Update edge features according to $S$;
8      if $j = dummy$ then
9          Terminate the selection;
10     else
11         $S \leftarrow S \cup \{j\}$;
```

Training. We train the model using the same labeled data with NCP. The overall training procedure follows Section 3 with $\gamma = 1$ except for the loss function. We employ the MSE loss function because it performs better than BCE loss and positive weighted BCE loss for the one-shot decoding model.

Algorithm 2: CapacityCut with the one-shot decoding

```
Input: Augmented graph $G$
Output: Selected nodes set $S$
1  Predict node selection probability $\{p_i\}_{i \in V_C} \leftarrow f_\theta(G)$;
2  if training then
3      Update $\theta \leftarrow \nabla \mathcal{L}(\{p_i\}_{i \in V}, \{\hat{y}_i\}_{i \in V})$;
4  else
5      Set the depot probability $p_0 \leftarrow 0, y_0 \leftarrow 0$;
6      $y_i \leftarrow round(p_i), \forall i \in V_C$;
7      Include $i$ to $S$ if $y_i = 1$;
```

C Further Experiments

C.1 Experimental results with limited wall clock time

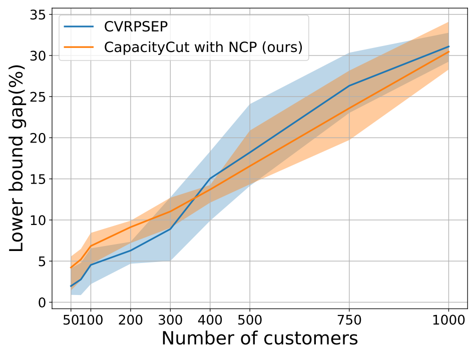
We compare performances of NCP and CVRPSEP with limited wall clock time (2 hours) instead of the limited number of subroutine calls. Experiments are conducted on the server equipped with Intel(R) Core(TM) i9-7980XE CPU. We evaluate the models only using CPU, as CVRPSEP is not implemented to use GPU acceleration. As shown in Table 3, the lower bound with NCP is tighter than CVRPSEP when the number of nodes is bigger than or equal to 400. Fig. 8 shows that NCP performs better in the large-scale CVRP.

C.2 Experiments on X-instances in CVRPLIB

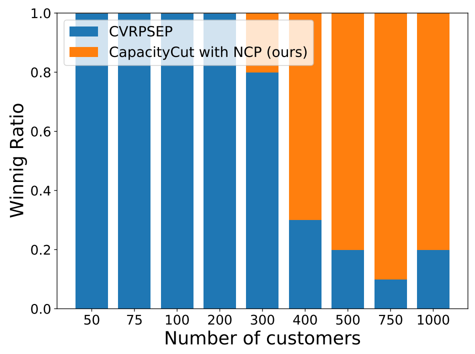
We apply the trained model to X-instances [30] in CVRPLIB without additional training to verify the transferability. Note that the training data is generated using uniform demand distribution $U(1, 100)$ but the X-instances have seven different demand distributions. Table 4 shows that the model performs better when the number of customers exceeds 200. NCP achieves a more significant LB improvement per iteration even when the demand distribution is different with the training demand distribution. Fig. 9 illustrates the lower bound gap with HGS algorithm. The results suggest that NCP mostly outperforms CVRPSEP in the large-scale problems with limited iterations.

Table 3: The mean and standard deviation of lower bound with limited wall clock time.

Method	Size	Mean LB	Std. LB	Avg. # of Calls	Avg. Δ LB
CVRPSEP	50	**9,363.570**	2,490.674	71.9	37.026
	75	**13,355.372**	7,289.166	114.4	35.059
	100	**15,937.238**	5,155.842	161.0	34.341
	200	**21,377.666**	4,762.775	341.0	23.097
	300	**31,288.780**	11,646.043	370.9	41.276
	400	42,315.017	12,517.111	226.7	91.894
	500	49,081.463	23,271.637	205.6	97.201
	750	67,719.115	21,080.550	211.7	158.350
	1,000	65,106.924	12,136.281	157.7	182.185
CapacityCut with NCP (ours)	50	9,146.218	2,422.141	42.1	56.076
	75	13,063.580	7,294.335	63.4	58.578
	100	15,577.428	5,146.759	96.5	54.625
	200	20,751.042	4,743.903	134.1	55.851
	300	30,804.764	12,187.478	113.6	119.368
	400	**43,341.285**	13,859.847	85.4	271.026
	500	**50,052.446**	23,024.014	62.3	446.744
	750	**69,964.785**	20,731.845	31.8	1,228.709
	1,000	**65,580.619**	11,257.778	19.1	1,626.640



(a) The range of LB gap for each test size.

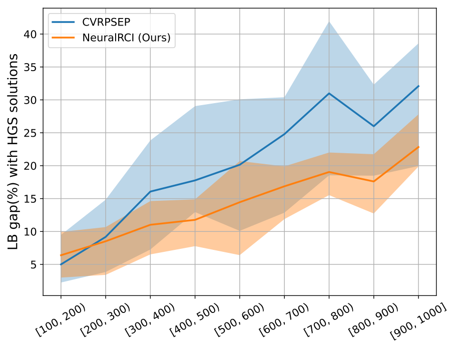


(b) The winning ratio out of 10 instances.

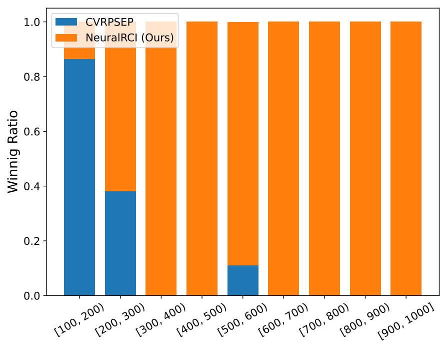
Figure 8: The results of the cutting plane method with NCP and CVRPSEP.

Table 4: The mean and standard deviation of lower bound of X-instances.

Method	Range	Mean LB	Std. LB	Avg. Iterations	Avg. Δ LB
CVRPSEP	[100, 200)	**25,769.830**	14,081.872	187.05 ($\leq$ 200)	48.625
	[200, 300)	36,994.622	25,865.421	200 ($\leq$ 200)	73.770
	[300, 400)	46,417.015	33,962.062	100 ($\leq$ 100)	177.987
	[400, 500)	61,982.074	48,613.320	100 ($\leq$ 100)	198.625
	[500, 600)	69,884.441	36,139.920	50 ($\leq$ 50)	459.653
	[600, 700)	63,584.630	26,472.213	50 ($\leq$ 50)	422.451
	[700, 800)	62,401.834	29,730.288	50 ($\leq$ 50)	433.237
	[800, 900)	82,898.842	40,405.746	50 ($\leq$ 50)	463.897
	[900, 1000]	105,572.352	89,510.664	50 ($\leq$ 50)	753.173
CapacityCut with NCP (ours)	[100, 200)	25,514.464	14,192.554	136.32 ($\leq$ 200)	60.119
	[200, 300)	**37,660.482**	26,874.093	167.95 ($\leq$ 200)	84.577
	[300, 400)	**50,172.212**	37,826.764	100 ($\leq$ 100)	215.539
	[400, 500)	**66,723.129**	51,968.514	100 ($\leq$ 100)	246.035
	[500, 600)	**76,593.828**	42,385.466	50 ($\leq$ 50)	593.841
	[600, 700)	**70,233.887**	28,931.966	50 ($\leq$ 50)	555.436
	[700, 800)	**71,564.764**	28,957.874	50 ($\leq$ 50)	616.495
	[800, 900)	**92,101.115**	44,707.908	50 ($\leq$ 50)	647.942
	[900, 1000]	**115,312.115**	85,135.955	50 ($\leq$ 50)	947.968



(a) The range of the LB gap in X-instances.



(b) Winning ratio between NCP and CVRPSEP in X-instances.

Figure 9: The results of the cutting plane method with NCP and CVRPSEP in X-instances.