Appendix: On the Misalignment in Non-Autoregressive GNNs for Neural Combinatorial Optimization

Jerry Sun[1], Zhixiao Xiong[2], Chi-Guhn Lee[1], and Eldan Cohen[1]

[1] University of Toronto, Toronto, Canada
{jhsun, cglee, ecohen}@mie.utoronto.ca
[2] Tsinghua University, Beijing, China
xiong-zx21@mails.tsinghua.edu.cn

Abstract. Neural combinatorial optimization (NCO) has emerged as a paradigm that leverages machine learning to tackle combinatorial problems, with Graph Neural Networks (GNNs) proving particularly effective for solution construction. In this work, through experiments with representative GNN-based non-autoregressive (NAR) methods— EfficientTSP and Difusco, we reveal a fundamental misalignment in the supervised NAR framework: reducing loss over the training batches does not correlate with lower optimality gap. We hypothesize that this misalignment arises from NAR methods' use of a static probability map which fails to adapt to previous decisions and account for variable dependencies, and present experimental results that support this hypothesis. To address this misalignment, we design a GNN-based autoregressive (AR) framework that leverages conditional dependencies through partial solution completion. Our empirical results demonstrate that this AR framework eliminates the observed misalignment. Furthermore, when compared to state-of-the-art NAR methods, it achieves comparable performance on in-distribution instances and shows superior generalization capabilities, while requiring far fewer labeled optimal solutions (10,000 vs 1-1.5 million).

Keywords: Graph Neural Networks, Neural Combinatorial Optimization

A Dataset Generation

To generate the MVC and MIS instances used in this study, we generate Erdős–Rényi (ER) graphs [18] using the NetworkX library [24]. Each n node ER graph is generated with edges established between node pairs with probability $p = 0.05 \pm 0.02$.

For uniformly distributed TSP instances, we generate Waxman graphs [58] using NetworkX [24] with the parameter $\beta = 1$ and domain coordinates ranging from $(0, 0)$ to $(1, 1)$ (unit square). In this setup, each node is assigned a coordinate within the unit square, resulting in a complete graph where every pair of nodes is connected. Edge costs are calculated based on the Euclidean distance between

Algorithm 1 Training Process for Proposed AR Framework

Require: Training instances $\{g_i, \hat{x}_i\}_{i=1}^{N}$, where $g_i = (V_i, E_i)$ is a graph and $\hat{x}_i$ is the set of variables true in optimal solution
Ensure: Trained GNN Model $\mathcal{M}$
Initialize empty training set $T \leftarrow \emptyset$
for each training instance $(g_i, \hat{x}_i)$ **do**
  Sample a partial solution $x_i \subset \hat{x}_i$ with size uniformly sampled from $\{1, ..., |\hat{x}_i|\}$
  Define the remainder of the solution $\tilde{x}_i' = \hat{x}_i \setminus \tilde{x}_i$
  for each decision variable $v \in g_i$ **do**
    if $v \in \tilde{x}_i$ **then**
      Set binary feature $f_{g_i}(v) \leftarrow 1$ ▷ Indicates inclusion in $\tilde{x}_i$
    else
      Set binary feature $f_{g_i}(v) \leftarrow 0$
    end if
    if $v \in \tilde{x}_i'$ **then**
      Assign label $l_{g_i}(v) \leftarrow 1$ ▷ Indicates v should be selected next
    else
      Assign label $l_{g_i}(v) \leftarrow 0$
    end if
  end for
  Update training set $T \leftarrow T \cup (g_i, \{f_{g_i}(v)\}_{v \in G_i}, \{l_{g_i}(v)\}_{v \in G_i})$
  Repeat the above process k times to sample k partial solutions
end for
Train model $\mathcal{M}$ on training set T by minimizing the cross-entropy loss
return Trained GNN Model $\mathcal{M}$

node coordinates. To make the TSP graphs more computationally tractable, we sparsify them by connecting each node only to its $k = 20$ nearest neighbors. This graph generation process, including sparsification, follows the conventions established in prior TSP studies [27, 50, 20].

To generate the ground-truth labels, we obtain an optimal solution for each problem instance using Gurobi [23].

B Pseudocode for Training Process of Proposed AR Framework

Algorithm 1 is the pseudocode for the training process of the proposed framework.

C Model Configurations

In this section, we detail the configurations and hyperparameters of the models used in our experiments. All models were implemented using the Deep Graph Library (DGL) [57]. For definitions and additional information on specific hyperparameters, please refer to the DGL documentation.

C.1 Non-Autoregressive Models

This subsection provides the configurations for the NAR models. For TSP, follow the original configurations from [27] and [50]. For MIS and MVC, the following hyperparameters are shared across all NAR models:

- Batch size: 64
- Number of layers: 4
- Hidden dimension: 128
- Batch normalization: Enabled
- Residual connections: Enabled

The specific configurations for each NAR model are:

- GAT: Number of heads = 2
- GatedGCN: N/A
- GCN: N/A
- GIN: Apply function layers = 2; Learn ϵ = True; Aggregation type = Max
- MoNet: Aggregation type = Max; Pseudo-dimension = 2; Number of kernels = 1
- GraphSage: Aggregation type = Pool

C.2 Proposed Autoregressive Models

This subsection provides the configurations for the AR models. For TSP, the configurations are provided in the main text. For MIS and MVC, the following hyperparameters are shared across all AR models:

- Batch size: 64
- Number of layers: 4
- Hidden dimension: 128
- Batch normalization: Enabled
- Residual connections: Enabled
- Number of partial solutions sampled: 100

The specific configurations for each AR model are:

- GAT: Number of heads = 2
- GatedGCN: N/A
- GCN: N/A
- GIN: Apply function layers = 2; Learn ϵ = True; Aggregation type = Max
- MoNet: Aggregation type = Max; Pseudo-dimension = 2; Number of kernels = 1
- GraphSage: Aggregation type = Pool

D Metrics

We use optimality gap as the primary metric for solution quality, defined as:

$$\text{Optimality Gap} = \frac{|z_{\text{pred}} - z_{\text{opt}}|}{z_{\text{pred}}} \times 100\% \tag{1}$$

where z_{pred} and z_{opt} represent the objective values of the constructed solution and the optimal solution, respectively. The quality of the generated probability maps is evaluated using the cross-entropy loss metric, the loss function used to train the models:

$$l_{\text{CE}} = -\frac{1}{|D_g|} \sum_{i \in D_g} y_i \log(\omega_i) + (1 - y_i) \log(1 - \omega_i) \tag{2}$$

where y_i is the ground-truth label (1 if variable i is in the optimal solution, 0 otherwise) and ω_i is the predicted probability for variable i.

E Corruption Experiments

This section provides the full procedure used to compare the robustness of NAR and AR frameworks under controlled corruption of probability maps. Our experiment focused on the TSP problem and involved systematically introducing errors into perfect probability maps. For the NAR framework, we first constructed a perfect probability maps for each problem instance using a single optimal solution, where edges in the optimal solution received very high probabilities (0.9-1.0) and non-optimal edges received very low probabilities (0-0.1). We then corrupted $p\%$ of values by replacing them with random samples from $[0, 1]$. For the AR framework, we followed an iterative approach: starting with an empty solution, at each step we generated a perfect probability map based on the optimal solution that included the current partial solution. We then applied the same corruption process, replacing $p\%$ of the probability values with random samples from $[0, 1]$, and used this corrupted map to select the next edge for inclusion in the solution.

F Additional Experiments

F.1 Detailed Analysis of AR vs NAR Frameworks

The results in Figure 1 provide evidence that the misalignment issue is inherent to the NAR framework rather than specific to TSP or any particular architecture. Across six different GNN architectures and two additional problems (MIS and MVC), we observe a similar pattern: while training losses steadily decrease (dotted lines), indicating improvement in probability map quality, the optimality gaps (solid lines) show no consistent improvement.

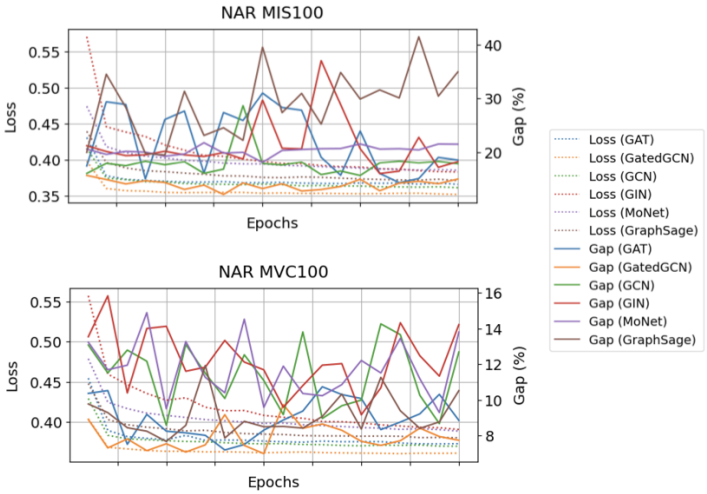

Fig. 1: Training loss versus optimality gap throughout the training process for MIS (left) and MVC (right) problems across six different GNN architectures using greedy decoding. For all architectures and both problems, dotted lines represent training loss while solid lines show optimality gaps. Lower is better for both metrics.

The results in Figure 2 demonstrate that the alignment displayed by our AR framework is consistent across different GNN architectures and two other problems (MIS and MVC).

Figure 3 illustrates the comparative performance between the NAR and AR frameworks across six GNN architectures on TSP100. The NAR framework (left) exhibits a notable misalignment between the training loss and optimality gap metrics. In contrast, the AR framework (right) demonstrates consistent alignment between these metrics across all architectural variants.

Figure 4 compares the optimality gaps between the NAR and AR models on three optimization problems: TSP100, MIS100, and MVC100. The results show that AR models consistently achieve lower optimality gaps across all problems. This consistent superior performance suggests that the AR framework's ability to condition decisions on previous variable assignments leads to more effective solution construction compared to the NAR framework's use of static probability maps. Please refer to Appendix C for the configurations of the models used in this experiment.

F.2 Batch-by-Batch Analysis

To further investigate the relationship between training loss and solution quality, we conducted a batch-by-batch analysis of EFFICIENTTSP on TSP20. Figure 5

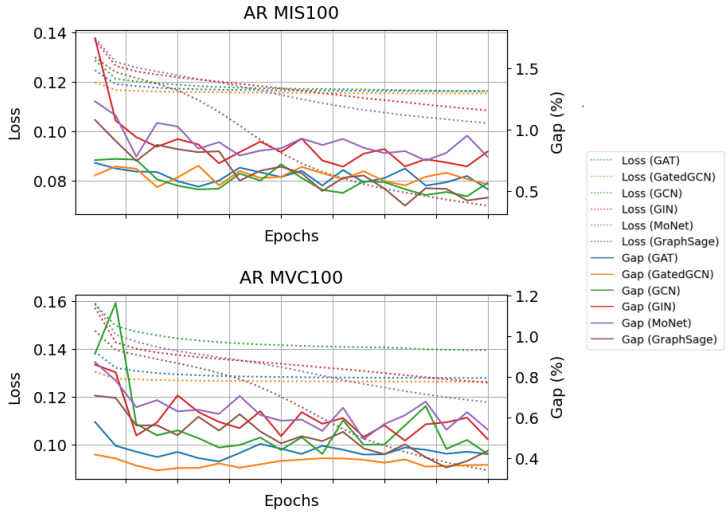

Fig. 2: Training loss versus optimality gap throughout the training process for the AR framework on MIS (left) and MVC (right) problems across six different GNN architectures using greedy decoding. For all architectures and both problems, dotted lines represent training loss while solid lines show optimality gaps. Lower is better for both metrics.

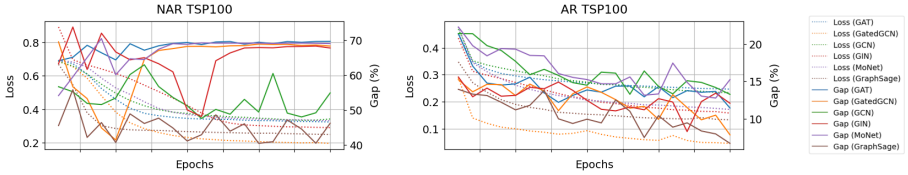

Fig. 3: Training loss versus optimality gap throughout the training process for the NAR framework (left) and the AR framework (right) on TSP100 across six different GNN architectures using greedy decoding. For all architectures and both frameworks, dotted lines represent training loss while solid lines show optimality gaps. Lower is better for both metrics.

plots the training loss versus optimality gap for individual batches. The results demonstrate that the model successfully learns useful inductive biases for solution construction, though most of this learning only occurs within the first epoch.

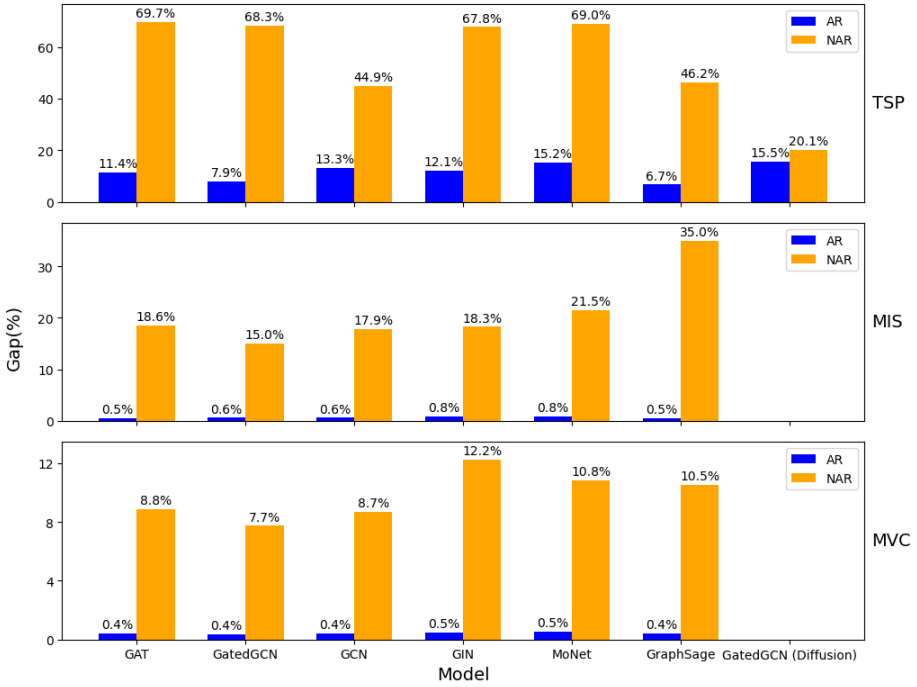

Fig. 4: Results comparing the optimality gap (%) achieved by the NAR models and the AR models on TSP100, MIS100, and MVC100. Lower is better.

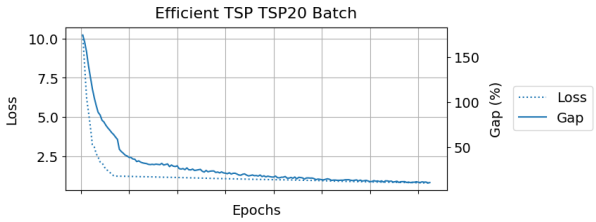

Fig. 5: Batch-by-batch analysis of training loss versus optimality gap for NAR and AR frameworks on TSP50. Each point represents a single batch, with the x-axis showing the training loss and y-axis showing the optimality gap achieved on that batch.

F.3 Comparison with Other Neural Combinatorial Optimization Methods

While our main analysis focuses on comparing AR and NAR approaches within GNN-based methods to highlight their fundamental differences, we compare our method against other NCO solution construction approaches in the appendix for completeness. Due to the wide variety of advanced search algorithms used across

these methods and the difficulty in establishing equivalent settings among them, we report only the greedy search results for this comparison. Some methods are excluded from this comparison because they do not report results on TSP50 or TSP100 instances or because they do not report results for greedy search. The results are presented in Table 1.

Table 1: Results against existing state-of-the-art NCO methods. All methods use greedy search. TSP50 results are trained on TSP50 instances and TSP100 results are trained on TSP100 instances. All baseline results are sourced from their respective papers.

ALGORITHM	TSP50 GAP % ↓	TSP100 GAP % ↓
OURS SL	0.35	1.1
DEITSP SL [56]	**0.12**	0.63
IMAGE DIFFUSION SL [22]	1.28	2.19
POINTER NETWORK SL [54]	11.4	-
LEHD SL [39]	-	**0.58**
TRANSFORMER NETWORK DRL [9]	0.31	-
AM DRL [46]	1.76	4.53
NCORL DRL [4]	4.54	5.15
POMO DRL [34]	0.64	1.07
EAN DRL [15]	2.23	-
S2VDQN DRL [28]	5.81	-

F.4 Runtime Analysis

In this section, we analyze the runtime performance of the proposed AR framework compared to the NAR framework. As expected, the AR framework requires more time to construct solutions due to the additional inference steps involved. Table 2 presents the experimental results on the average runtime per problem instance for all models used in the paper. It is worth noting that the AR framework can be adjusted to add multiple nodes or edges per iteration, although this modification was not implemented in this paper. Such an adjustment would introduce a trade-off between solution quality and runtime efficiency, which could be explored in future work.

G Relation to Guided Tree Search

Guided Tree Search [36] is a supervised method for constructing solutions to the MIS problem. It employs a tree search algorithm that leverages probability maps generated by a Graph Convolutional Network (GCN) [31]. The training

Table 2: Average runtime (in seconds) per problem instance. All problems have 100 nodes. All experiments performed on the same hardware.

Framework	Runtime s ↓		
	Tsp	Mis	Mvc
Nar	8.8	0.31	0.16
Ar	42.9	0.56	0.27

process is the same to that of the NAR framework we examine in Section **??**, with a slight difference: the GCN produces multiple probability maps for each graph instance in a single forward pass. Specifically, each node is assigned scores m times (with $m = m$ in their configuration), representing its likelihood of inclusion in 32 different potential solutions. This approach acknowledges the presence of multiple optimal solutions for each MIS instance, allowing the model to generate a diverse set of focused probability maps.

The GCN is trained using a hindsight cross-entropy loss:

$$\mathcal{L}_{\text{hindsight}} = \min_{i=1,\ldots,m} \mathcal{L}_{\text{CE}}\left(y_{\text{true}}, f_i\right) \tag{3}$$

where $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy loss, y_{true} are the true labels, and $f_1, ..., f_m$ are the m different probability maps produced by the GCN. The training aims to minimize the lowest cross-entropy loss among all generated maps. A tree search algorithm is then employed to construct a feasible solution from these maps, switching between different maps as needed. During the traversal of a single probability map, the method iteratively includes the node with the highest probability into the solution and marks all its neighbors as excluded. After this step, all marked nodes, both included and excluded, are removed from the graph. The GCN then generates a new probability map for the remaining subgraph. This process repeats until all nodes have been marked.

We classify this method as non-autoregressive. However, it differs from other NAR methods because it conceptually resembles conditional generation in autoregressive methods. To condition the probabilities on the existing partial solution, this method does not explicitly train the GCN to generate conditioned outputs. Instead, it reduces the original graph after each selection by removing the selected node and its neighbors, and feeds this reduced subgraph back into the GCN to produce a new probability map. This is a unique property of the MIS problem, where the state of the current partial solution can be implicitly represented by the modified input graph.

Formally, given a graph $G = (V, E)$ and an independent set of nodes $K \subseteq V$, let $K' = K \cup \bigcup_{v \in K} N(v)$, where $N(v)$ denotes the set of neighbors of node v. Let $G' = G[V \setminus K']$ be the induced subgraph obtained by removing K' from G. If L is a maximum independent set on $\mathcal{G}'$, then $K \cup L$ constitutes the largest independent set on G that includes K.

In other words, finding the largest independent set on G conditioned on the partial solution K is equivalent to finding the maximum independent set on the reduced subgraph G'. In G', all nodes in K and their neighbors, along with all edges incident to them, have been removed. This reduction effectively encodes the condition into the graph structure. This property suggests that while the model does not explicitly perform conditional generation, it achieves a similar outcome by manipulating the input graph to reflect the current partial solution. Moreover, this approach of implicit conditional generation through input manipulation does not readily extend to other combinatorial optimization problems. As such, we classify this method as non-autoregressive because the GCN model itself is not trained to generate predictions autoregressively.

References

1. Ahn, S., Seo, Y., Shin, J.: Learning what to defer for maximum independent sets. In: International conference on machine learning. pp. 134–144. PMLR (2020)
2. Amizadeh, S., Matusevych, S., Weimer, M.: Learning to solve circuit-sat: An unsupervised differentiable approach. In: International Conference on Learning Representations (2019)
3. Beck, J.C., Fox, M.S.: Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. Artificial Intelligence **117**(1), 31–81 (2000)
4. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
5. Berto, F., Hua, C., Zepeda, N.G., Hottung, A., Wouda, N., Lan, L., Park, J., Tierney, K., Park, J.: RouteFinder: Towards Foundation Models for Vehicle Routing Problems. Transactions on Machine Learning Research (2025), https://openreview.net/forum?id=QzGLoaOPiY
6. Bertsimas, D., Dunn, J.: Machine learning under a modern optimization lens. Dynamic Ideas LLC Charlestown, MA (2019)
7. Böther, M., Kißig, O., Taraz, M., Cohen, S., Seidel, K., Friedrich, T.: What's wrong with deep learning in tree search for combinatorial optimization. In: International Conference on Learning Representations (2022)
8. Bresson, X., Laurent, T.: Residual gated graph convnets. arXiv preprint arXiv:1711.07553 (2017)
9. Bresson, X., Laurent, T.: The transformer network for the traveling salesman problem. arXiv preprint arXiv:2103.03012 (2021)
10. Brucker, P.: Classification of scheduling problems. Scheduling Algorithms pp. 1–10 (2004)
11. Cappart, Q., Goutierre, E., Bergman, D., Rousseau, L.M.: Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 1443–1451 (2019)
12. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 3677–3687 (2021)
13. Chu, G., Stuckey, P.J.: Learning value heuristics for constraint programming. In: Integration of AI and OR Techniques in Constraint Programming: 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings 12. pp. 108–123. Springer (2015)

14. Dechter, R., Meiri, I.: Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. Artificial Intelligence **68**(2), 211–241 (1994)
15. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the tsp by policy gradient. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15. pp. 170–181. Springer (2018)
16. Drakulic, D., Michel, S., Andreoli, J.: Goal: A generalist combinatorial optimization agent learner. In: Proceedings of the International Conference on Learning Representations (ICLR) (2025)
17. Drori, I., Kharkar, A., Sickinger, W.R., Kates, B., Ma, Q., Ge, S., Dolev, E., Dietrich, B., Williamson, D.P., Udell, M.: Learning to solve combinatorial optimization problems on real-world graphs in linear time. In: 2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA). pp. 19–24. IEEE (2020)
18. Erdos, P., Rényi, A., et al.: On the evolution of random graphs. Publ. math. inst. hung. acad. sci **5**(1), 17–60 (1960)
19. Fang, H., Song, Z., Weng, P., Ban, Y.: Invit: a generalizable routing problem solver with invariant nested view transformer. In: Proceedings of the 41st International Conference on Machine Learning. pp. 12973–12992 (2024)
20. Fu, Z.H., Qiu, K.B., Zha, H.: Generalize a small pre-trained model to arbitrarily large tsp instances. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 7474–7482 (2021)
21. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. Advances in neural information processing systems **32** (2019)
22. Graikos, A., Malkin, N., Jojic, N., Samaras, D.: Diffusion models as plug-and-play priors. Advances in Neural Information Processing Systems **35**, 14715–14728 (2022)
23. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2023), https://www.gurobi.com
24. Hagberg, A., Swart, P., Schult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
25. Huang, J., Sun, Z., Yang, Y.: Accelerating diffusion-based combinatorial optimization solvers by progressive distillation. arXiv preprint arXiv:2308.06644 (2023)
26. Joshi, C.K., Cappart, Q., Rousseau, L.M., Laurent, T.: Learning the travelling salesperson problem requires rethinking generalization. Constraints **27**(1), 70–98 (2022)
27. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv preprint arXiv:1906.01227 (2019)
28. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. Advances in neural information processing systems **30** (2017)
29. Khalil, E.B., Morris, C., Lodi, A.: Mip-gnn: A data-driven framework for guiding combinatorial solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 10219–10227 (2022)
30. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations (ICLR) (2015), https://arxiv.org/abs/1412.6980

31. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2017)
32. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep policy dynamic programming for vehicle routing problems. In: International conference on integration of constraint programming, artificial intelligence, and operations research. pp. 190–213. Springer (2022)
33. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019)
34. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems **33**, 21188–21198 (2020)
35. Li, Y., Guo, J., Wang, R., Yan, J.: T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. Advances in Neural Information Processing Systems **36** (2023)
36. Li, Z., Chen, Q., Koltun, V.: Combinatorial optimization with graph convolutional networks and guided tree search. Advances in neural information processing systems **31** (2018)
37. Lombardi, M., Gualandi, S.: A lagrangian propagator for artificial neural networks in constraint programming. Constraints **21**, 435–462 (2016)
38. Lu, T., Ma, S., Tao, C.: Empowering targeted neighborhood search via hyper tour for large-scale tsp. arXiv preprint arXiv:2510.20169 (2025)
39. Luo, F., Lin, X., Liu, F., Zhang, Q., Wang, Z.: Neural combinatorial optimization with heavy decoder: Toward large scale generalization. Advances in Neural Information Processing Systems **36** (2023)
40. Luo, F., Lin, X., Wu, Y., Wang, Z., Xialiang, T., Yuan, M., Zhang, Q.: Boosting neural combinatorial optimization for large-scale vehicle routing problems. In: The Thirteenth International Conference on Learning Representations (2025)
41. Marty, T., Boisvert, L., François, T., Tessier, P., Gautier, L., Rousseau, L.M., Cappart, Q.: Learning and fine-tuning a generic value-selection heuristic inside a constraint programming solver. Constraints **29**(3), 234–260 (2024)
42. Mechqrane, Y., Bessiere, C., Elabbassi, I.: Using large language models to improve query-based constraint acquisition. In: IJCAI 2024-33rd International Joint Conference on Artificial Intelligence. pp. 1916–1925 (2024)
43. Min, Y., Bai, Y., Gomes, C.P.: Unsupervised learning for solving the travelling salesman problem. Advances in Neural Information Processing Systems **36** (2023)
44. Min, Y., Gomes, C.P.: On size and hardness generalization in unsupervised learning for the travelling salesman problem. arXiv preprint arXiv:2403.20212 (2024)
45. Ouyang, W., Li, S., Ma, Y., Wu, C.: Learning to segment for vehicle routing problems. arXiv preprint arXiv:2507.01037 (2025)
46. Peng, B., Wang, J., Zhang, Z.: A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In: Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11. pp. 636–650. Springer (2020)
47. Qiu, R., Sun, Z., Yang, Y.: Dimes: A differentiable meta solver for combinatorial optimization problems. Advances in Neural Information Processing Systems **35**, 25531–25546 (2022)
48. Smith, B.M., Grant, S.A.: Succeed-first or fail-first: A case study in variable and value ordering. Technical Report (1996)
49. Smith, B.M., Grant, S.A.: Trying harder to fail first. In Proceedings of ECAI'98 (1998)

50. Sun, Z., Yang, Y.: Difusco: Graph-based diffusion solvers for combinatorial optimization. Advances in Neural Information Processing Systems **36** (2023)
51. Tassone, J., Choudhury, S.: A comprehensive survey on the ambulance routing and location problems. arXiv preprint arXiv:2001.05288 (2020)
52. Toenshoff, J., Ritzert, M., Wolf, H., Grohe, M.: Graph neural networks for maximum constraint satisfaction. Frontiers in artificial intelligence **3**, 580607 (2021)
53. Verhaeghe, H., Cappart, Q., Pesant, G., Quimper, C.G.: Learning precedences for scheduling problems with graph neural networks. In: 30th International Conference on Principles and Practice of Constraint Programming (CP 2024). pp. 30–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2024)
54. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Advances in neural information processing systems **28** (2015)
55. Wang, H.P., Wu, N., Yang, H., Hao, C., Li, P.: Unsupervised learning for combinatorial optimization with principled objective relaxation. Advances in Neural Information Processing Systems **35**, 31444–31458 (2022)
56. Wang, M., Zhou, Y., Cao, Z., Xiao, Y., Wu, X., Pang, W., Jiang, Y., Yang, H., Zhao, P., Li, Y.: An efficient diffusion-based non-autoregressive solver for traveling salesman problem. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2025)
57. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
58. Waxman, B.M.: Routing of multipoint connections. IEEE journal on selected areas in communications **6**(9), 1617–1622 (1988)
59. Xia, Y., Yang, X., Liu, Z., Liu, Z., Song, L., Bian, J.: Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In: Forty-first International Conference on Machine Learning (2024)
60. Xiao, Y., Wang, D., Li, B., Wang, M., Wu, X., Zhou, C., Zhou, Y.: Distilling autoregressive models to obtain high-performance non-autoregressive solvers for vehicle routing problems with faster inference speed. In: Proceedings of the AAAI conference on artificial intelligence. vol. 38, pp. 20274–20283 (2024)
61. Xiao, Z., Song, W., Chen, Q.: Dynamic resource allocation using virtual machines for cloud computing environment. IEEE transactions on parallel and distributed systems **24**(6), 1107–1117 (2012)
62. Xing, Z., Tu, S.: A graph neural network assisted monte carlo tree search approach to traveling salesman problem. IEEE Access **8**, 108418–108428 (2020)
63. Zhao, H., Yu, K., Huang, Y., Yi, R., Zhu, C., Xu, K.: Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. Graphical Models **101284** (2025)
64. Zhou, J., Wu, Y., Song, W., Cao, Z., Zhang, J.: Towards omni-generalizable neural methods for vehicle routing problems. In: International Conference on Machine Learning. pp. 42769–42789. PMLR (2023)
65. Zhou, S., Ding, Y., Zhang, C., Cao, Z., Jin, Y.: Dualopt: A dual divide-and-optimize algorithm for the large-scale traveling salesman problem. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39 (2025)