SOLVING DIVERSE COMBINATORIAL OPTIMIZATION PROBLEMS WITH A UNIFIED MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Combinatorial Optimization (CO) covers a wide range of problems that exist in many real-world scenarios, while solving them using learning based methods has drawn great attention. Developing a unified deep model to solve diverse CO problems has many benefits, including a reduction in the need for hand-crafted designs for individual problems and enhanced flexibility for few-shot learning in unseen problem types. Meanwhile, a unified model with a single architecture and parameter set for diverse CO problems remains absent. To the best of our knowledge, we are the first to formally investigate and develop such a unified model. Motivated by the success of the next-token-prediction concept, we formulate each solution into an Markov Decision Process, and train the model with transformer backbone using tokenized data collected from problem solution trajectories. However, directly training the unified model is challenging due to the long token length of the trajectories, which arises from the complex observation space of CO problems, resulting from their NP-hard nature. Furthermore, using the same model to simultaneously predict observations and actions—distinct types of elements within a trajectory—further increases training difficulty. To address these challenges, we introduce two key designs. First, to reduce token length, we implement a CO-prefix design that aggregates the static features of the problems. Second, to account for the heterogeneity of state and action tokens within the MDP, we adopt a two-stage self-supervised learning scheme. In the first stage, a dynamic prediction model is learned, which then serves as a pre-trained model for subsequent policy generation. Experiments across a set of nine problems demonstrate the robust problem-solving capabilities of our unified model, along with its few-shot and even zero-shot generalization abilities. We believe our framework provides a valuable complement to existing neural CO methods that focus on achieving optimal performance for individual CO problems.

1 INTRODUCTION

Combinatorial optimization (CO) problems are crucial in a wide range of real-world scenarios, such as logistics, industrial management, etc (Singh & Rizwanullah, 2022). To solve these generally NP-hard problems, traditional integer programming and heuristics have been widely studied to obtain either exact or near optimal solutions in the past decades. With the rapid growth of deep learning, solving CO problems using learning based methods has drawn increasing attention and led to the rising field of Neural Combinatorial Optimization (NCO) (Kim et al., 2022; Drakulic et al., 2024). Depending on the solution generation scheme, NCO methods can be generally classified into auto-regressive ones and the non-autoregressive ones, where the former are more favored in recent literature (Bello et al., 2016; Kool et al., 2018; Kwon et al., 2020; Kim et al., 2022). The auto-regressive methods incrementally construct solutions, where the complete problem-solving process can be naturally viewed as a Markov Decision Process (MDP). These end-to-end methods offer significant computational efficiency and are more flexible in generating feasible solutions, as they can easily avoid constraint-violating actions within the MDP framework (Kim et al., 2022).

However, a significant limitation still remains: models from existing literature can only deal with specific problem types, lacking the capability to handle diverse problems simultaneously. There are significant benefits to use one unified model across diverse problems. First, it reduces the need of hand-crafted designs of each individual domain. Second, it can be utilized to unseen problem

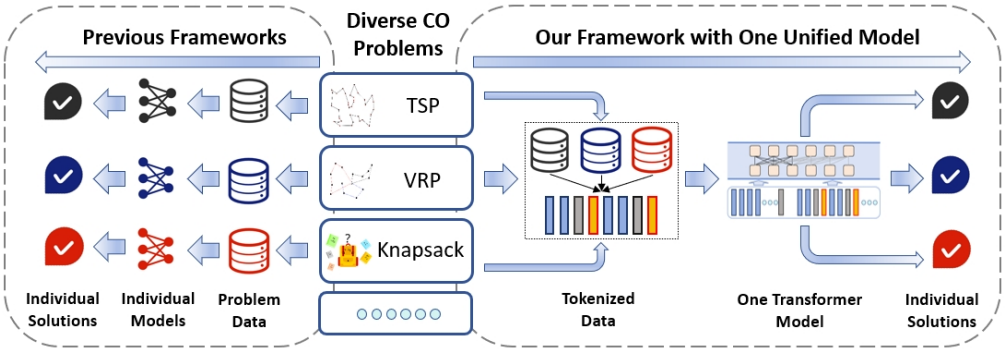


Figure 2: The difference between previous frameworks and ours to solve diverse CO problems. While previous frameworks require individual models with specific designs to adapt to different problems, our framework only utilizes one unified model.

types much easier and faster than specific models trained from scratch, due to its potential few-shot ability. Even though some literature claimed to propose generic frameworks, they only use the same general architecture across different problems with specific model structures and different learning parameters, which leads to the loss of generality, as shown in Figure 2. The development of these NCO methods aligns with the famous No Free Lunch Theorem (NFLT) (Wolpert & Macready, 1997), which states that the only way one strategy, i.e. a deep model in NCO, can outperform another if it is specialized to the specific problem structures under considerations. Most literature avoids the challenge of achieving generality across different problems, and focuses on improving performances on individual ones, illustrated as Model A and Model B in Figure 1. Thus we directly explore such a challenge where few tackled in NCO, and a new research problem emerges: *Can we develop a unified model with one neural architecture and parameter set that solves diverse CO problems simultaneously, with strong few-shot capability?*

Recently, the concept of next-token-prediction has marked a new era in general artificial intelligence, excelling in processing data across multiple scenarios, domains, and even modalities. The most successful examples are the large language models (LLMs) and multimodal large languange models (MLLMs) (Achiam et al., 2023; Dubey et al., 2024), which can generalize across various natural language process (NLP) and computer vision (CV) scenarios and excel in few-shot learning tasks. Furthermore, the concept has also been applied to decision-making tasks directly (Chen et al., 2021). For instance, Reed et al. (2022) developed a generalist agent to handle different control environments simultaneously, such as Atari games and robot benchmarks. Motivated by these breakthroughs, we explore whether a unified model can be designed to tackle diverse CO problems under the same next-token-prediction framework.

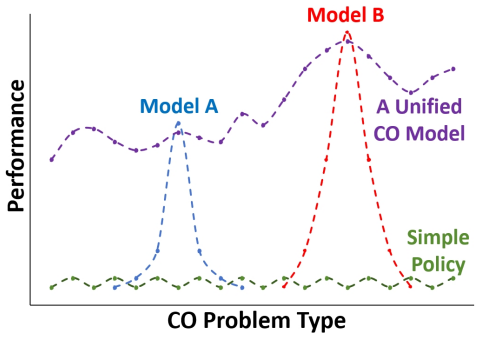


Figure 1: The No Free Lunch Theorem of optimization.

Generally, we collect solutions of raw problem instances generated by state-of-the-art solvers from a mix of problem sources. Following the commonly adopted auto-regressive MDP formulation of existing literature, we generate optimization trajectories where actions are iteratively selected to based on temporary partial solutions. These trajectories are serialized into flat token sequences, which are trained using a single transformer backbone, as shown in Figure 2. However, directly applying existing training schemes to CO problems is often inefficient. Since most CO problems are NP-hard, the observation space can be vast, leading to long token sequences and low training efficiency. Additionally, a full trajectory consists of different types of elements, such as observations and actions. Predicting all elements without considering their individual roles further increases the overall training difficulty.

To tackle these challenges, we introduce two approaches to improve generic training performances considering the common characteristics of CO problems. First, we propose a non-causal, decoder-only architecture that incorporates a CO-prefix to reduce the overall token length. Unlike other environments where observations in an MDP can be fully dynamic, most information in a CO problem comes from its static description data. For instance, in a Traveling Salesman Problem (TSP), the distances between node pairs remain unchanged regardless of the visiting order. Therefore, we utilize a CO-prefix to aggregate the problems' static features, while the subsequent main trajectory handles dynamic observations. This reduces token length and improves training efficiency. Second, we decompose the entire token generation process into two self-supervised learning stages to reduce training difficulty. In the first stage, the model focuses solely on learning to predict forward dynamics, which then serves as the pre-trained model for the subsequent policy generation. These two stages are designed to handle the heterogeneous elements within the trajectory, thereby reducing the overall training difficulty.

We should point out that even though one recent literature declared to achieve multi-task learning with cross-problem generalization for vehicle routing problems (VRP), it cannot be further generalized to a unified model that we aim to develop (Liu et al., 2024). They formulate VRPs as different combinations of shared attributes, including capacity, backhauls, time windows, duration limit and open route. The single model designed to solve VRPs within the attribute combinations still heavily relies on human-crafted designs, and fails in generalizing to problems beyond these specific combinations. In contrast, the framework we propose can be applied to any CO problems where a feasible solution can be formulated as an MDP.

To summarize, our key contributions are:

- To the best of our knowledge, we are the first to thoroughly investigate solving diverse CO problems using a single unified model and to present a corresponding framework. We believe that our framework provides a valuable complement to existing NCO methods that focus on achieving optimal performance for individual CO problems.
- To address the challenges of directly applying existing next-token prediction concepts to CO problems, we introduce a CO-prefix design and a two-stage self-supervised learning scheme to reduce token length and training difficulty.
- We establish a comprehensive testbed featuring nine CO problems to evaluate the generic problem-solving ability of our unified CO model. Experiments demonstrate that the model achieves strong generic problem-solving capabilities with only a slight reduction in performance. Additionally, we showcase its few-shot and even zero-shot generalization abilities when tackling new problems.

2 Related Works

2.1 Learning based Methods for CO problems

Research on NCO can be broadly divided into two categories: non-autoregressive and autoregressive approaches.

Regarding non-autoregressive approaches, many methods directly train reinforcement learning (RL) models to guide operators in refining feasible solutions. This scheme involves selecting an operator from a candidate pool (Lu et al., 2019) or determining where an operator should be applied (Chen & Tian, 2019; Ma et al., 2021; Wu et al., 2021). These methods aim to imitate and improve upon traditional heuristic search strategies through a data-driven approach. Another trend in non-autoregressive methods is to learn an intermediate problem representation, which is then used to guide the solution search. For example, a heatmap can be trained to predict the adjacency matrix for TSP problems based on expert solutions, followed by a search to find the final solution (Joshi et al., 2019; Fu et al., 2021). In addition to using deep neural networks to directly predict the adjacency matrix, diffusion models and probabilistic methods can also generate intermediate representations (Sun & Yang, 2023; Karalias & Loukas, 2020). These approaches are particularly effective in adapting to larger problem scales. However, these methods are often limited on specifc problem types since they lack flexibility in handling complex problem constraints.

As for autoregressive methods, they aim to incrementally build a solution by selecting new nodes step by step, ultimately constructing a complete feasible solution. The pioneering work in this area was the Pointer Network, which was first tested on the TSP(Vinyals et al., 2015). Subsequent research combined the idea with reinforcement learning (RL), demonstrating its effectiveness across a wider range of CO problems(Bello et al., 2016). Routing problems, a significant subclass of CO problems, have been extensively studied within this autoregressive framework using RL (Kool et al., 2018; Kwon et al., 2020). To account for both node- and edge-level features, a matrix-encoding framework was later developed (Kwon et al., 2021). These methods are particularly advantageous due to their fast inference speed, as the computational complexity during testing remains low. The are also much more flexible in generating feasible actions handling various problem constraints.

A recent trend in the field is the exploration of algorithm generalization capabilities. Existing methods focus on generalizing across different data distributions (Zhou et al., 2023; Bi et al., 2022) and problem scales (Zong et al., 2022; Li et al., 2021). Regarding generalization to multiple tasks, one study attempts to solve various VRPs by decomposing them into several elementary tasks (Liu et al., 2024). However, this decomposition heavily relies on human-designed rules, limiting its generalization potential. As far as we know, there is no architecture for a truly general-purpose unified model capable of addressing diverse CO problems.

2.2 Next-Token-Prediction in Decision-Making

Besides the significant success of the next-token-prediction in both LLMs and MLLMs, researchers have also investigated how to incorporate it with decision-making problems directly.

Chen et al. (2021) first studied using Transformer (Vaswani, 2017) as an effective backbone to handle various control environments in an offline RL manner, including Atari, Open AI Gym, etc. They train a single policy model to generate actions at each decision step. Janner et al. (2021) further proposed the trajectory transformer that predicts all elements within a trajectory. Besides offline RL, the similar architecture has also been incorporated with imitation learning (Reed et al., 2022; Shafiullah et al., 2022; Brohan et al., 2022; Zhou et al., 2022). One notable application following this line of research is the Generalist Agent, known as GATO (Reed et al., 2022), which successfully extended its capabilities to multiple control environments using a unified model. Wen et al. (2022) further implemented the GATO structure, referred to as DB1, and extended it to solve TSP problems. Building on these successes, it is natural to consider transformers as the backbone for a unified model capable of addressing diverse CO problems.

However, we found that Wen et al. (2022) employed an individual pretrained GCN model (Kipf & Welling, 2016) specialized to TSP, which was used to generate TSP state embeddings. These embeddings were then used to train the unified model, rather than using the original TSP data. We believe this approach contradicts the original concept of a unified model, which relies solely on a single architecture and parameter set. Nevertheless, we adopt the unified model structure proposed by GATO and re-implemented in DB1 as an important baseline for comparison, where only the original trajectory data is processed.

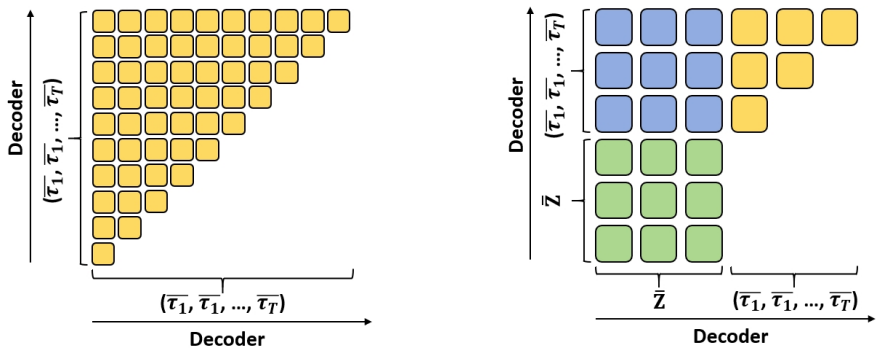
3 Methodology

3.1 Preliminaries

3.1.1 Auto-regressive MDP Formulation for CO

We first formulate the sequential construction process of a CO problem solution as an MDP. Suggested by existing auto-regressive NCO methods (Zhang et al., 2023), a complete solution can be constructed by incrementally constructed via multiple decision steps.

Let $\mathcal{S}$ denote the entire state space with states $s_t \in \mathcal{S}$ and $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ with actions $a_t \in \mathcal{A}$ as the action space. All states are assumed to be reachable from the original state s_1. Since a CO problem is fully observed and deterministic, the transition from a state s_t to s_{t+1} is fully determined by the action a_t. Each state s_t is represented as a set of actions taken before. A *policy* on the MDP refers to a distribution $P(s'|s)$ over the states s' reachable from s via a single action. A feasible CO problem



(a) Causal decoder-only architecture without CO prefix.

(b) Non-causal decoder-only architecture with CO-prefix.

Figure 3: Two architecture designs for the unified model. a) Causal decoder-only architecture without CO prefix, where each token is only conditioned on the past tokens and only trajectory data is processed, adopted in Reed et al. (2022). The entire token length is large. b) Non-causal decoder-only architecture with CO-prefix, where tokens in the CO-prefix shares richer representations conditioned on both prior and past tokens. The trajectory no longer process duplicated static information.

solution represented as a complete trajectory τ can be further induced by the policy over T steps via $\prod_{t=1}^{T} P(s_{t+1}|s_t)$.

We note that most CO problems exhibit a common property of tail recursion: after applying a series of construction steps to an instance, the remaining tail subproblem itself becomes an instance of the original CO problem, as discussed in Drakulic et al. (2024). Any problem with this tail-recursion property can be formulated as the aforementioned MDP. In this paper, we focus on CO problems with such a property.

3.1.2 Trajectory Datasets

To prepare the trajectory datasets for training, we first obtain the final optimized solutions from state-of-the-art solvers for various problem types. We then trace their complete optimization MDPs, $\tau = (\tau_1, \tau_2, ..., \tau_T)$, considering states and actions, where $\tau_t = (s_t, a_t)$.

To jointly handle diverse features from different problems and distributions, we then tokenize all trajectory elements within the trajectory data via one same tokenizer. Specifically, discrete values, such as the node indexes, are flattened into sequences of integers within the range of $[Min_d, Max_d)$. Continuous values, such as demands and positions, are first encoded to $[Min_c, Max_c]$ if not already in the range, and then discretized to N_{bin} uniform bins. The final trajectory token sequence $\overline{\tau}$ is formulated with observation tokens followed by an action spliter token I_a, then action tokens, as shown in Figure 4:

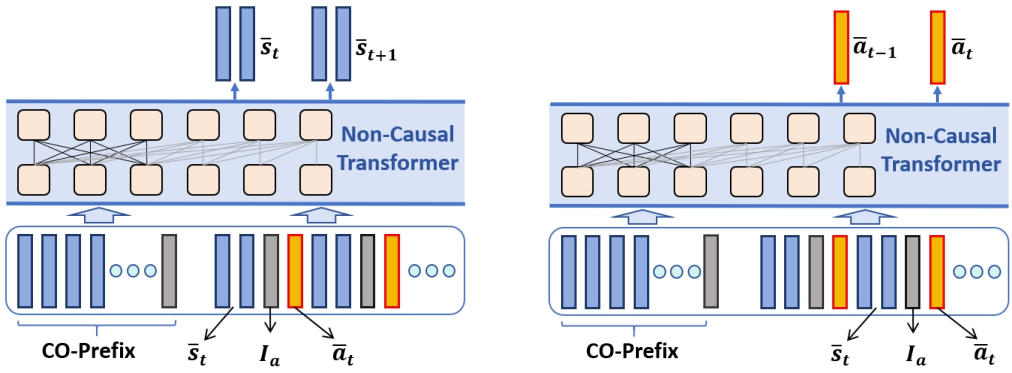
$$\overline{\tau} = (\overline{\tau_1}, \overline{\tau_2}, ..., \overline{\tau_T}), \quad \text{where } \overline{\tau_t} = (\overline{s_t}, I_a, \overline{a_t}) \tag{1}$$

Note that the length of a fully tokenized sequence can sometimes be excessively long. To address this, we use selected contiguous segments from the full trajectory. Furthermore, we only preserve dynamic observations in the intermediate progress within s_t. The static information of the raw problem instances is aggregated within a CO-prefix design, as introduced in the following.

3.2 Non-causal Transformer with CO-prefix

Due to the NP-hard complexity of most CO problems, the observation space and dimensionality can be substantial, leading to long token sequences and reduced training efficiency.

To tackle this challenge, we first decompose the original state representation into static and dynamic ones, since most information in a CO problem comes from its static description data. For instance, static information of a given TSP instance includes the positions of each individual city, which remains unchanged through an optimization MDP. While the dynamics only include the current position. We further employ a CO-prefix design to capture static information, which is prepended to



(a) The dynamics forward stage. (b) The policy generation stage.

Figure 4: Two-stage self-supervised learning to train the unified CO model.

the beginning of the token trajectory. The subsequent sequence then focuses solely on dynamic observations. This approach allows us to avoid duplicating observation representations by tokenizing only the current dynamic state instead of the entire sequence, as done in previous literature. Such a design greatly reduce token length and improves training efficiency. Let Z and $\overline{Z}$ denote raw and tokenized CO-prefix, the final token sequence fed into the model is $(\overline{Z}; I_p, \overline{\tau})$, where I_p denotes a spliter token in between.

Although the sequential nature of Markov Decision Processes (MDPs) with time-dependent ordering makes the causal transformer architecture a natural choice due to its simple and effective one-directional design as suggested in previous sequential decision-making literature (Chen et al., 2021; Reed et al., 2022), as shown in Figure 3(b), it has certain limitations. In particular, the CO-prefix Z is time-invariant since it only contains static representations, where each token within $\overline{Z}$ should be fully visible and processed with each other in a bi-directional manner.

To address this, we adopt a non-causal transformer architecture where the CO-prefix tokens are processed bi-directionally, ensuring comprehensive context integration, while the remainder of the sequence is still handled in a one-directional manner, as shown in Figure 3(a). Tokens in the CO-prefix shares richer representations conditioned on both prior and past tokens and thus improves the overall performances.

Action and CO-prefix Mask To ensure that each action selected by the unified model is feasible during inference, the output policy must be masked to filter out actions that violate problem constraints, using the action mask provided by the problem environment.

It's important to note that during the generation of trajectory data, action masks are collected alongside the trajectory data at each step. During training, the action mask is transformed into the CO-prefix mask, with each token corresponding to infeasible actions being masked in the attention module. For example, in the Traveling Salesman Problem (TSP), the CO-prefix mask encompasses all coordinates of already visited cities. In the case of the Flexible Flow Shop Problem (FFSP), it refers to the job duration entries of completed tasks. This design enables the model to focus on more relevant tokens for feasible actions without increasing the overall token length."

3.3 Two-Stage Self-Supervised Learning

Since a complete trajectory consists of different types of elements, such as observations and actions, predicting them without distinguishing their individual roles further increases the training difficulty.

To tackle the challenge above, we decompose the entire token generation process into two stages in a self-supervised learning manner, including a dynamics forward stage and a policy generation stage as follows.

- **Dynamics forward stage.** In the first stage, we pre-train the model to predict the next observation given the current action. The training loss for a training batch $\mathcal{B}$ is defined as follows:

$$\mathcal{L}(\theta, \mathcal{B}) = -\sum_{b=1}^{|B|}\sum_{t=1}^{T^b} \log p_\theta(\overline{s_{t+1}^b}|(\overline{Z^b}, I_p, \overline{\tau_1^b}, \overline{\tau_2^b}, ..., \overline{\tau_t^b})), \tag{2}$$

  where T^b is the amount of trajectory units in the current token length. Since MDP transitions are deterministic in CO problems, ths dynamics model can be accurately trained with the same amount of data.

- **Policy generation stage.** In the second stage, we fine-tune the model to generate actions based on the pretrained model in advance. The training loss for a training batch $\mathcal{B}$ is defined as follows:

$$\mathcal{L}(\theta, \mathcal{B}) = -\sum_{b=1}^{|B|}\sum_{t=1}^{T^b} \log p_\theta(\overline{a_{t+1}^b}|(\overline{Z^b}, I_p, \overline{\tau_1^b}, \overline{\tau_2^b}, ..., \overline{\tau_t^b}, \overline{s_{t+1}^b}, I_a)) \tag{3}$$

This two-stage decomposition simplifies the learning process by decomposing the overall process into two sub-tasks, allowing the model to first understand intermediate dynamics and then generate qualified policy. This leads to faster and more effective convergence during training.

Table 1: The summary of the evaluated CO problems, along with individual expert solver to collect trajectories, the token length in the CO-prefix and the token length in observation per step. N denotes either node, item or job amounts. M denotes for machine amount in FFSP.

Problem	Expert Solver	Prefix-Token	Obs-Token
TSP	LKH3 (Helsgaun, 2017)	$2N$	$N+2$
VRP	LKH3 (Helsgaun, 2017)	$3N+2$	$N+3$
OP	Gurobi (Gurobi Optimization, 2018)	$3N+2$	$N+4$
PCTSP	ILS [1]	$4N+2$	$N+3$
SPCTSP	re-opt with ILS	$4N+2$	$N+3$
Knapsack	dynamic programming	$2N$	$N+1$
ATSP	LKH3 (Helsgaun, 2017)	$N \times N$	N
MIS	Kamis Lamm et al. (2017)	$N \times N$	N
FFSP	MatNet (Kwon et al., 2021)	$N \times M$	$M+1$

4 Performance Evaluation

4.1 Evaluation Protocols

4.1.1 Problem and Expert Selection

To evaluate the generic problem solving ability of our proposed framework, we construct a problem set from 9 diverse domains for evaluation.

We first select four common routing problems that were deeply investigated in recent literature (Kool et al., 2018; Kim et al., 2022), including Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), Orienteering Problem (OP) and Prize Collecting TSP (PCTSP). Stochastic PCTSP (SPCTSP) is further added to show how our model deals with uncertainty, suggested by (Kool et al., 2018). We also consider the Asymmetric TSP (ATSP), where the problem is defined on adjacency matrix without Cartesian coordinates (Kwon et al., 2021). Beyond routing problems, we first evaluate our model on the Knapsack problem, following previous literature Bello et al. (2016); Grinsztajn et al. (2023). Maximum Independent Set (MIS) is adopted as a representation problem that mainly leverages features from graph structure Sun & Yang (2023), in which information carried by each token is extremely sparse. Finally, we evaluate our model on the flexible flow shop problem (Kwon et al., 2020).

For each problem, the trajectories are collected from individual expert solver, shown in Table 1. We set the problem scale as $N = 20$, either for node, item or job amounts. We align the instance

Table 2: The overall performance comparison on nine problems.

	TSP				CVRP			
Method	Obj.	Gap	Score	Time	Obj.	Gap	Score	Time
Random	10.47	-	0.00%	9s	13.25	-	0.00%	29s
Expert	3.84	0.00%	100.00%	2h	6.11	0.00%	100.00%	5h
GATO/DB1	3.99	3.80%	97.68%	1h	**6.63**	**8.51%**	**92.72%**	**2h**
Ours-DR	3.88	1.04%	99.40%	8m	6.75	10.47%	91.04%	15m
Ours	**3.87**	**0.78%**	**99.55%**	**9m**	6.66	9.00%	92.30%	16m
	PCTSP				OP			
Method	Obj.	Gap	Score	Time	Obj.	Gap	Score	Time
Random	9.25	-	0.00%	20s	1.93	-	0.00%	8s
Expert	3.16	0.00%	100.00%	2h	5.38	0.00%	100.00%	1h
GATO/DB1	3.27	3.48%	98.19%	1h	4.91	8.87%	85.46%	53m
Ours-DR	3.27	3.48%	98.19%	13m	5.00	7.06%	88.99%	8m
Ours	**3.20**	**1.27%**	**99.34%**	**13m**	**5.06**	**5.95%**	**90.72%**	**8m**
	SPCTSP				Knapsack			
Method	Obj.	Gap	Score	Time	Obj.	Gap	Score	Time
Random	9.24	-	0.00%	20s	38.14	-	0.00%	6s
Expert	3.31	0.00%	100.00%	2h	63.89	0.00%	100.00%	10m
GATO/DB1	3.30	0.30%	100.17%	1h	**62.19**	**2.66%**	**93.40%**	**35m**
Ours-DR	3.28	-0.09%	100.51%	13m	61.78	3.30%	91.81%	4m
Ours	**3.26**	**-1.51%**	**100.84%**	**13m**	61.99	2.97%	92.62%	4m
	ATSP				MIS			
Method	Obj.	Gap	Score	Time	Obj.	Gap	Score	Time
Random	10.49	-	0.00%	10s	9.11	-	0.00%	7m
Expert	3.85	0.00%	100.00%	2h	10.44	0.00%	100.00%	7m
GATO/DB1	-	-	-	-	-	-	-	-
Ours-DR	4.38	13.76%	91.87%	9m	10.35	0.86%	93.23%	11m
Ours	**4.22**	**9.61%**	**94.43%**	**10m**	**10.35**	**0.86%**	**93.23%**	**10m**
	FFSP							
Method	Obj.	Gap	Score	Time				
Random	45.00	-	0.00%	12min				
Expert	27.31	0.00%	100.00%	5m				
GATO/DB1	-	-	-	-				
Ours-DR	29.20	6.92%	89.32%	29min				
Ours	**29.10**	**6.55%**	**89.88%**	**27min**				

generation scheme with previous literature for each problem, and the details can be referred in Appendix A.2.

4.1.2 Evaluation Protocols[2]

Hyperparameters During training, for every epoch we process 400 batches of 128 instances, which are sampled from a mixed set of all 9 problems. The total token length of each episode is 1000, either tailored or padded from the trajectory data with prefix. We use 10 layers in the transformer architecture with embedding dimension of 768. For tokenization, we set the discrete range, continuous range and bin number as $[200,)$, $[0, 4]$ and 1800 respectively. We evaluate the model on the validation dataset every two epochs and apply early stopping, terminating training if no improvement is observed for 6 consecutive epochs. During inference, we evaluate the performances on each problem individually, with a test dataset of 10000 each. We report the absolute objective value, the inference time cost and the percentile gap to the expert performance. We list more implementation details in A.1 for reproducibility.

Metrics We report four metrics respectively. Following previous CO related literature (Kool et al., 2018), we report the original objective and the gap from the expert results, and the evaluation time

[2]Our code is available at https://anonymous.4open.science/r/uniCO-35CC/

on the entire test dataset. Following the literature that studies generic decision-making (Reed et al., 2022), we also report the performance score as a percentage, where 100% corresponds to the per-task expert and 0% to a random policy. It is calculated as $Score = |obj_e - obj_r|/|obj - obj_r|$, where obj_e and obj_r denotes the objective of expert and a random policy respectively (Wen et al., 2022).

Ablation and Baselines We evaluate our proposed model with two variations, either with or without the two-stage supervised learning. We use *Ours-DR* to denote the model DRectly trained to generate actions, as shown in Table 2. As for the baseline methods, we first demonstrate corresponding expert approach of each problem as a straight-forward comparison. We further compare with the GATO framework (Reed et al., 2022), which was also re-implemented and reported by Wen et al. (2022) as DB1. Note that we also implement the original GATO framework from details in the original paper manually, since it is not open-sourced. Instead of using any prefix or pretrain schemes, GATO is trained using a causal transformer structure, where the trajectory data of each problem is prepended by a prompt sequence from the same problem. The prompt consists of multiple step transitions from other episodes. Other key hyperparameters remain the same as ours. Note that we found GATO/DB1 cannot converge on ATSP, MIS and FFSP under our evaluation setting, thus we remove the three problems for GATO/DB1 to train only across 6 problems, while we stick to 9. We analyze the reason and influence in the next section. To calculate the optimization score, we also report the performance of a random policy in terms of objective and time. The evaluation time of the random policy shows the environment time cost in our implementation.

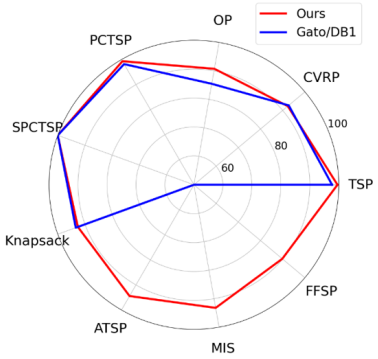


Figure 5: Performances on diverse problem types.

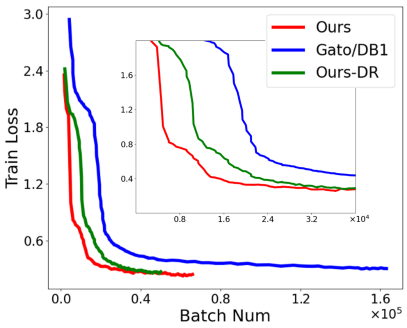


Figure 6: The training loss along with total batch used of three models.

4.2 Performances of Generic Problem Solving

The main results are illustrate in Table 2. Generally, our proposed unified model showcase its universal problem-solving ability across diverse CO problems. Except for the score of 89.88% on FFSP, it achieves scores of over 90% on all other problems.

Our CO-prefix design is critical and significant. While two variants of our unified model are trained on the entire nine problems, the GATO/DB1 are trained only on the former six problems, since we found that they can not converge on the latter three within a reasonable timeframe under our evaluation settings, shown in Figure 5. This limitation arises since that GATO/DB1, lacking a prefix design, computes full observation tokens at each step which makes them extremely inefficient when the observation space is large. For example, in both ATSP and MIS problems, the original static information is carried by the instance adjacency matrix, with a complexity of $O(N^2)$. In each training episode, GATO/DB1 can only process one or two complete trajectory steps with or without their prepended prompt sequences. The sparse loss signals generated on the action tokens impede the model's convergence. Even for the remaining problems, GATO/DB1 still converges much slower than ours across all problems, as shown in Figure 6. Moreover, our unified model design still shows superior performance on four problems out of six.

The two-stage self-supervised learning scheme greatly improves performances. Compared to a unified model that is directly trained to generate actions, a model fine-tuned on a pre-trained forward dynamics model demonstrates superior performance across all nine problems. For example, in the ATSP problem, the training scheme we introduced yields a performance improvement of 2.56

We further analyze the adaptability of our unified model to different CO problems. The individual training losses on each problem are illustrated in Figure 8(a). Since the model is trained on trajectory data from respective expert solvers, these solvers serve as natural performance upper bounds for our unified model. An exception is observed in SPCTSP, where uncertainty suggests the potential for performance beyond that bound. Generally, we identify two problem properties that present challenges for our unified model at this stage. The first is information sparsity in token sequences. Although ATSP and TSP share similar complexities from a heuristic perspective, our model can more easily learn the policy for solving TSP while struggling with ATSP. This difficulty arises because the token sequences for ATSP are much sparser than those for TSP, with problem data stored in the adjacency matrix rather than in the nodes. The sparser the token information, the harder it is for our model to learn effectively. The second challenge is the limitations imposed by constraints within the problems. OP and CVRP have significantly more constraints to satisfy compared to TSP, making the feasible action space under each state much more complex to learn. This complexity further restricts the final optimization quality.

4.3 Performances on Few-shot Ability

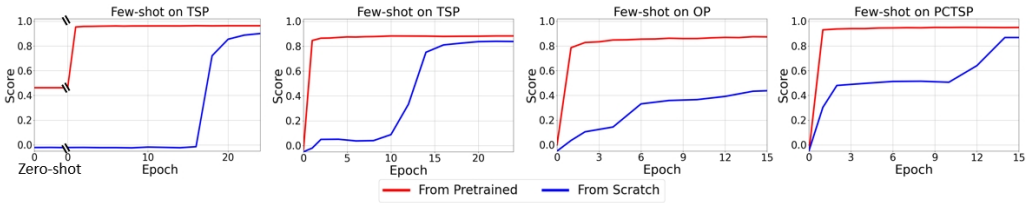


Figure 7: The few-shot results on four routing problems.

To evaluate the few-shot capability of our model on unseen problems, we selected four routing problems and trained a total of four unified models. Each model was trained on three out of the four problems in a leave-one-out manner and was gradually fine-tuned using data from the fourth unseen problem. We report the optimization scores for the new problem and compare them with those of a model trained from scratch on the corresponding problem.

Overall, we found that our model demonstrates few-shot learning across all four problem settings with limited data. In each case, the model achieves high solution quality from the very first epoch. This is a significant advantage of a unified CO model, as CO encompasses a wide range of problems with diverse settings. Consequently, the pre-trained unified model can be quickly adapted to an unseen problem with minimal data, eliminating the need to retrain a separate model. This offers enhanced convenience and efficiency in many real-world scenarios.

Furthermore, in addition to its few-shot capabilities, we observed zero-shot performance on TSP. Since TSP serves as a foundational version of many routing problem variants, our model, pre-trained on the other three problems, can directly generate semi-optimized solutions without any additional data for fine-tuning.

5 Future Works

In this paper, we thoroughly investigate how to develop a unified model to solve diverse CO problems simultaneously, and evaluate the performance of our proposed unified model implementation in 9 problems. We believe that our approach provides a valuable complement to existing NCO methods that focus on achieving optimal performance for individual CO problems.

As for our future work, to overcome the performance loss caused by token information sparsity as discussed in Section 4.2, one promising direction is to incorporate our current transformer backbone with GNN based structures, since many CO problems are defined or can be defined based on graphs. However, how to maintain the universal token processing ability remains challenging. Another direction is to incorporate our model with other recent progress in auto-regressive NCO methods, such as Kwon et al. (2020); Chalumeau et al. (2023).

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems*, 35:31226–31238, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Felix Chalumeau, Shikha Surana, Clément Bonnet, Nathan Grinsztajn, Arnu Pretorius, Alexandre Laterre, and Tom Barrett. Combinatorial optimization with policy adaptation using latent space search. *Advances in Neural Information Processing Systems*, 36:7947–7959, 2023.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in neural information processing systems*, 32, 2019.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Paul Erd6s and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Bruce L Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:48485–48509, 2023.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018. URL `http://www.gurobi.com`.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33: 6659–6672, 2020.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.

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

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

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.

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.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. Heuristics*, 23(4):207–229, 2017. doi: 10.1007/s10732-017-9337-x. URL https://doi.org/10.1007/s10732-017-9337-x.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Fei Liu, Xi Lin, Zhenkun Wang, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1898–1908, 2024.

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

Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34:11096–11107, 2021.

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.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

Guman Singh and Mohammad Rizwanullah. Combinatorial optimization of supply chain networks: A retrospective & literature review. *Materials today: proceedings*, 62:1636–1642, 2022.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:3706–3731, 2023.

Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Ying Wen, Ziyu Wan, Ming Zhou, Shufang Hou, Zhe Cao, Chenyang Le, Jingxiao Chen, Zheng Tian, Weinan Zhang, and Jun Wang. On realization of intelligent decision-making in the real world: A foundation decision model perspective. *arXiv preprint arXiv:2212.12669*, 2022.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems*, 36:11952–11969, 2023.

Allan Zhou, Vikash Kumar, Chelsea Finn, and Aravind Rajeswaran. Policy architectures for compositional generalization in control. *arXiv preprint arXiv:2203.05960*, 2022.

Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, pp. 42769–42789. PMLR, 2023.

Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. Rbg: Hierarchically solving large-scale routing problems in logistic systems via reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4648–4658, 2022.

A APPENDIX

A.1 EVALUATION DETAILS.

In this section ,we provide more implementation details for reproducibility.

Table 3: Implementation details.

Module	Element	Detail
System	OS	Ubuntu 22.04.2
	CUDA	11.7
	Python	3.11.4
	Pytorch	2.0.1
	Device	2*NVIDIA A100 80G
Hyperparameters	Backbone	Llama
	Embedding dimension	768
	Layer Num	10
	Q Head Num	8
	KV Head Num	8
	Max token length	1000
	RMS Norm epsilon	1e-6
	Weight Decay	1e-4
	Early Stopping Runs	6
	M of μ-law	4
	μ of μ-law	15
	Shaped Discrete Token Range	[0, 200)
	Shaped Continuous Token Range	[200, 2000)
	Batch Size	1200
	Batch Num	15000
	Optimizer	AdamW
	inital learning rate	0
	max learning rate	2.5e-4
	leanring rate warmup ratio	5%
	leanring rate decay ratio	75%
	leanring rate decay factor	10
	leanring rate decay style	cosine

A.2 PROBLEM DETAILS.

In this section, we continue to introduce the implementation details on each CO problem. We use N to denote either node, item or job amount, and M to denote the total machine amount in FFSP.

A.2.1 TRAVELING SALESMAN PROBLEM (TSP)

In the TSP, one should find the shortest route that visits each city exactly once and returns to the starting city. The objective is to minimize the total distance of the tour.

Data Generation. We implement the dataset generation scheme described by Kool et al. (2018), for all TSP instances, the positions of N nodes are uniformly randomly sampled in unit square. The expert trajectory is collected by the well adopted state-of-the-art solver LKH3 (Helsgaun, 2017).

Token Design For prefix tokens, we include the original Cartesian coordinates of each city with a total number of $2N$ as continuous values. For tokens per step, we only record the continuous coordinates of the current city as observation to reduce the sequence length. To help with convergence, the information about which cities have been visited is embodied by the dynamic prefix attention mask. Specifically, when generating actions, the coordinates of current and previously visited cities in prefix sequence are ignored.

A.2.2 VEHICLE ROUTING PROBLEM (VRP)

In the Capacitated VRP (Toth & Vigo, 2014), each city has a certain demand. One should construct multiple routes with minimal a distance that all start and end at a given depot, where the total demands of cities within one route should not exceed the capacity limit. Except for the depot, each city should be visited exactly once.

Data Generation. We implement the dataset described by Nazari et al. (2018). The expert trajectory is also collected via LKH3 (Helsgaun, 2017). Specifically, each city $i \in \{1, 2, .., N\}$ has a demand $0 < \delta_i \leq D$, where $D > 0$ is the capacity of the vehicle (route). For each route R_j, the total demand of the cities along cannot exceed the vehicle's capacity, i.e. $\sum_{i \in R_j} \delta_i \leq D$. For our experiments, We random sample the location coordinates of the depot and the cities within the unit square uniformly. The capacity is set to $D = 20$ and the discrete demands are sampled uniformly from $\{1, 2, ..., 9\}$.

Token Design For prefix tokens, we include the original Cartesian coordinates of the depot and each city, as well as the discrete demands of the cities, with a total number of $3N + 2$ continuous or discrete values. For tokens per step, we record the capacity left and the location coordinates of the vehicle currently as observation. To help with convergence, the prefix tokens about cities whose demand has been met will be ignored by the dynamic prefix attention mask when generating actions.

A.2.3 ORIENTEERING PROBLEM (OP)

In the OP (Golden et al., 1987), each node is assigned with a specific prize. One should construct a single tour that maximize the sum of prizes, starting and ending at a give depot. The tour does not have to include every node anymore, but need to be shorter than a length limit.

Data Generation. We implement the data generation scheme by Fischetti et al. (1998); Kool et al. (2018). The expert trajectory is collected via Gurobi (Gurobi Optimization, 2018). Specifically, The location coordinates of depot as well as N node are random sampled uniformly in the unit square. To make the problem more challenging, we made the prize p_i for each node i proportional to its distance from the depot by setting them as:

$$p_i = 1 + \left[99 \cdot \frac{d_{0i}}{\max_{j=1}^{n} d_{0j}} \right], \hat{p}_i = \frac{p_i}{100}$$

where d_{0i} is the distance from node i to the depot. As for the length limit of the route, we set the fixed max length as $T = 2$, which makes the optimal number of access nodes different from instance to instance

Token Design For prefix tokens, we include the original Cartesian coordinates of the depot and each node, as well as all of the node prizes, with a total number of $3N + 2$ continuous values. For tokens per step, we record the length left to the limit, current location coordinates, and the total prize we have gotten so far, with a total of 4 continuous values as observation. By utilizing the dynamic prefix mask, all prefix tokens related to nodes that have been visited will be ignored when generating actions, which helps the model converge efficiently.

A.2.4 PRIZE COLLECTING TSP (PCTSP)

In the PCTSP (Balas, 1989), the sum of total prize is no longer a optimization objective, but a constraint. One should minimize the total route length plus the sum of penalties of unvisited nodes which are given ahead, as well as collecting at least a minimal total prize.

Data Generation. We implement the data generation scheme by Kool et al. (2018). The expert trajectory is collected via an implementation of Iterated Local Search (ILS). Specifically, as the OP problem mentioned previously, the location coordinates of the depot and all nodes are randomly sampled uniformly within the unit square. For each node i, the associated prize p_i and penalty β_i need to be balanced carefully. If the penalty is too small, the choice of node is almost entirely determined by the total reward constraint; If the penalty is too large, all nodes are always accessed and the total reward constraint fails. Following the reference Kool et al. (2018), we set the prize and penalty as:

$$t_i \sim \text{Uniform}(0, 1), \quad \rho_i = t_i \cdot \frac{4}{N}$$

$$\beta_i \sim \text{Uniform}\left(0, 3 \cdot \frac{K^N}{N}\right)$$

where K^N is about half of the trajectory length of the TSP problem with N cities, we roughly set it as $K^{20} = 2$, and the minimum total prize is set to 1 for our experiments.

Token Design For prefix tokens, we include the original Cartesian coordinates of the depot and nodes, as well as the prize and penalty of each node, with a total number of $4N + 2$ continuous values. For tokens per step, we record the prize-to-go from the minimum total prize constraint and the location coordinates currently, with a total of 3 continuous values as observation. We also set the dynamic prefix mask to ignore all prefix tokens related to nodes visited before when generating actions, which helps the model converge efficiently.

A.2.5 STOCHASTIC PCTSP (SPCTSP)

In the SPCTSP, we show how our unified model performs when dealing with uncertainty. The expected prize of each node is known before the optimization starts, while the real collected prize can only be revealed after visitation. **Data Generation.** The data generation for SPCTSP is the sameas in PCTSP, except that we additionally generate the expected prize, which has the same distribution of the real prize.The expert solution algorithm is a modified version of ILS, where the tour is re-optimized iteratively, as suggested by Kool et al. (2018).

Token Design The token design remains the same as PCTSP, except that in prefix tokens the prize of each node is represented as the expected prize but not the real one.

A.2.6 ASYMMETRIC TSP (ATSP)

In the ATSP, the distances between node pairs are no longer determined by Euclidean distances based on node coordinates. Considering a directed graph, the distances are no longer necessarily the same in both directions, and are given in an asymmetric cost matrix upfront. We show how our model performs when dealing with features of $O(N^2)$ complexity. We follow the data generation scheme proposed by Kwon et al. (2021), and adopt LKH3 as the corresponding expert solver Helsgaun (2017). **Data Generation.** The data generation scheme remains the same as TSP, except that the coordinates are provided while only adjacency matrix is visible. **Token Design**

A.2.7 KNAPSACK

In the Knapsack problem, a group of items with specific values and volumes are given. The optimization objective is to maximize the total value of items selected without exceeding the total capacity. We designed the problem generation scheme manually and implemented the dynamic programming algorithm for trajectory collection.

Data Generation. We implement a manually designed data generation scheme. Specifically, The values v_i of each item $i \in \{1, 2, ..., N\}$ are randomly sampled as:

$$v_i \sim \text{Uniform}(2, N)$$

To make the problem more challenging, items of higher value should have a larger volume. We further introduce some randomness and set the volume k_i of item i as:

$$k_i = (1 + t)v_i$$

where $t \sim \text{Uniform}(\{-0.5, 0.5\})$, which means we increase or decrease the volume of item i uniformly and randomly.

Token Design For prefix tokens, we include the values and volumes of all N items, with a total number of $2N$ discrete values. For tokens per step, we only record the real-time left capacity as observation. By utilizing the dynamic prefix mask, all prefix tokens related to the item that has been packaged will be ignored when generating actions, which helps the model converge efficiently.

A.2.8 MAXIMUM INDEPENDENT SET (MIS)

In the MIS, an independent set is a set of vertices such that no two vertices in the set are adjacent. One should find the largest possible independent set in the graph, meaning it contains the most vertices among all possible independent sets.

Data Generation. We follow the random graph generation scheme proposed by Erd6s & Rényi (1960) , and directly implement the script provided by Sun & Yang (2023) to generate the graphs. The expert solver to generate trajectories is the Kamis Lamm et al. (2017) **Token Design** For prefix tokens, we directly use the binary adjacency matrix, with a total of N^2 tokens. For tokens per step, we record whether each node is selected, excluded or not decided yet, with a total of N tokens.

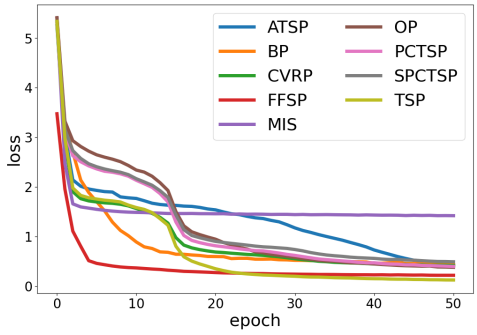
A.2.9 Flexible Flow Shop Problem (FFSP)

In the FFSP, N jobs have to be processed in S stages with the same order. Each job in each stage can be handled by a machine from M total machines. The time required for each job at different stages on different machines varies. Each machine can only process at most one job at the same time. The goal is to schedule all jobs so that they can be finished with a minimum of time. **Data Generation.** We directly adopt the data generation scheme and script provided by Kwon et al. (2021). We further and implement the corresponding MatNet as the only NCO expert solver in our experiments for trajectory generation.

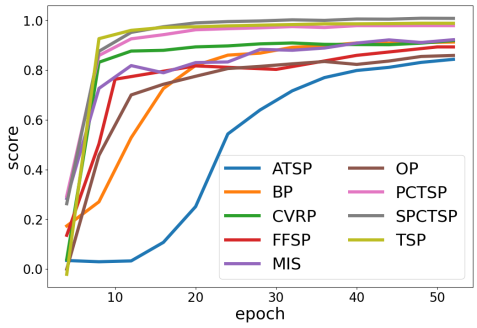
Token Design The prefix tokens include the job duration in each stage on the corresponding machine, with a total amount of $N \times M$. In each step, we track one single machine and decide either to make it wait or to assign it with a new job. The step token is the job duration related to the current machine, with a total length of M.

A.3 Learning Analysis

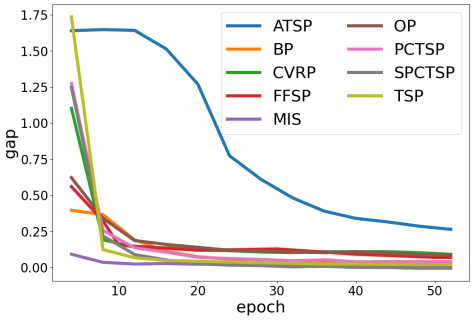
We list showcase the training loss, the evaluation gap and optimization score of each individual problem in our main results.



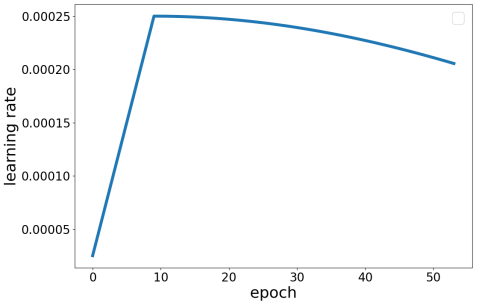
(a) The aggregated training loss of each problem, of which the weighted average is the total training loss shown in Fig 6.



(b) The score of each problem.



(c) The performance gap of each problem.



(d) The learning rate during training.