Solving NP-hard Min-max Routing Problems as Sequential Generation with Equity Context

Jiwoo Son[*1] **Minsu Kim**[*1] **Sanghyeok Choi**[2] **Hyeonah Kim**[1] **Jinkyoo Park**[1]
[1]Korea Advanced Institute of Science and Technology (KAIST)
[2]Seoul National University
{sonleave25, min-su, hyeonah_kim, jinkyoo.park}@kaist.ac.kr
{hyeok9855}@snu.ac.kr

Abstract

Min-max routing problems aim to minimize the maximum tour length among agents as they collaboratively visit all cities, i.e., the completion time. These problems include impactful real-world applications but are known as NP-hard. Existing methods are facing challenges, particularly in large-scale problems that require the coordination of numerous agents to cover thousands of cities. This paper proposes a new deep-learning framework to solve large-scale min-max routing problems. We model the simultaneous decision-making of multiple agents as a sequential generation process, allowing the utilization of scalable deep-learning models for sequential decision-making. In the sequentially approximated problem, we propose a scalable contextual Transformer model, Equity-Transformer, which generates sequential actions considering an equitable workload among other agents. The effectiveness of Equity-Transformer is demonstrated through its superior performance in two representative min-max routing tasks: the min-max multiple traveling salesman problem (min-max mTSP) and the min-max multiple pick-up and delivery problem (min-max mPDP). Notably, our method achieves significant reductions of runtime, approximately 335 times, and cost values of about 53% compared to a competitive heuristic (LKH3) in the case of 100 vehicles with 1,000 cities of mTSP.

1 Introduction

Routing problems are combinatorial optimization problems that are notoriously difficult to solve. The traveling salesman problem (TSP) and vehicle routing problems (VRPs) are representative problems where the objective is to determine the optimal or shortest tour route(s) for one or multiple agents, such as robots, vehicles, or drones. These problems are classified as NP-hard, posing significant challenges [1]. Various approaches have been proposed to solve (NP-hard) routing problems, including mathematical programming techniques that aim to achieve provable optimality (e.g., [2, 3]), task-specific heuristic solvers (e.g., [4, 5]), and deep learning-based methods that provide task-agnostic and fast heuristic solvers [6–30]. These methods have shown promising results even for large-scale problems which have more than 10,000 cities [3, 31].

Min-max routing problems are distinct from standard (min-sum) routing problems in that they focus on minimizing the cost of the most expensive route among multiple agents rather than minimizing the sum of cost of routes. While these problems are clearly less studied than min-sum routing problems due to their difficulty [32, 6], min-max routing problems are particularly relevant in time-critical applications such as disaster management [33], where minimizing completion time (or service time)

[*]equal contribution

ICML 2023 Workshop: Sampling and Optimization in Discrete Space (SODS 2023).

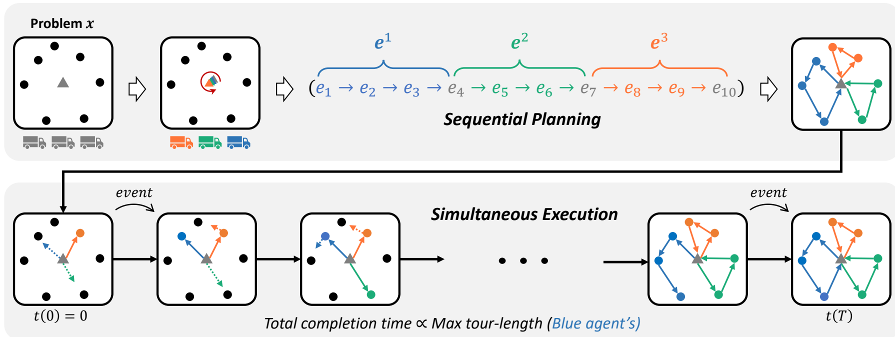


Figure 1: Illustration of sequential planning and real-time execution strategy of Equity-Transformer. The encoder produces *encoder representation*, and the decoder auto-regressive produces sequence. The sequence becomes the tour planning for each agent. The objective of generated sequence is maximum tour length (blue agent's tour length), which is directly proportional to total completion time on real-time execution.

is crucial. However, min-max routing is challenging as it should leverage multiple agents considering total completion time; the algorithm must consider proper cooperation between multi-agents and balanced assignment to minimize maximum tour length. Classical exact algorithms struggle to solve min-max routing problems [32], and powerful heuristic approaches for min-sum problems are not well generalized to the min-max case [34], particularly for large-scale problems [35], owing to the inherent differences between min-max and min-sum problems.

Recently, deep learning methods have been utilized to address min-max routing problems [36, 37, 6] as an alternative to classical approaches. Notably, representative min-max routing techniques such as ScheduleNet [6] and the decentralized attention network (DAN) [37] aim to handle the min-max nature by treating it as a simultaneous decision-making process with decentralized modeling of multiple agents. These approaches capture the intuitive real-time execution of decision-making in multi-agent min-max routing; they modeled even-based execution directly, i.e., they directly model simultaneous execution depicted at Fig. 1. Despite their intuitive decentralized modeling, the performances of these models are not competitive due to the difficulty of training decentralized cooperative policy using sparse and delayed reward [6].

We view that min-max routing problems can be effectively solved using a fully centralized approach, employing simple sequential generation techniques and harnessing the power of sequential models such as the Transformer [38] for streamlined modeling (see Fig. 1). To adapt the Transformer model specifically for min-max routing problems, additional modifications are necessary, and in this regard, our work introduces novelty.

To this end, we propose a new Transformer model, *Equity-Transformer*, for min-max routing problems. Equity-Transformer is built on top of the attention model (AM) [7], an effective transformer for min-sum routing problems. Our model contains two key inductive biases to solve multi-agent min-max routing problems:

- *Multi-agent positional encoding for order bias.* We introduce virtual orders on agents to model the simultaneous decision as sequences. To model the precedence of agents (i.e., order bias among agents), we add positional encoding of ordered agents and inject it into the encoder.
- *Context encoder for equity context.* To promote equitable tours for multiple agents, we incorporate an equity context into the sequence generators. Our approach entails considering crucial factors such as temporal tour length, the target tour length, and the desired number of cities to be visited, thereby enhancing the fairness of the generated tours.

Our method performs remarkably at the min-max routing problems, outperforming both existing classical heuristic and learning-based methods. As a highlight, Equity-Transformer achieves 334× speed improvement and 53% reduction of solution cost compared to the representative classical heuristic solver (LKH-3) when solving the mTSP with 1000 cities. Also, our method achieves 1217× faster speed and 9% reduction of solution cost than the representative learning-based method, the ScheduleNet [6] for the same problem.

2 Related Works

Our work aims to solve **min-max routing problems** using a Transformer-based **constructive solver**. In this section, we briefly explain the min-max and the min-sum routing problems, and provide an overview of two sub-categories of neural combinatorial optimization (NCO), the learning-based strategy for combinatorial optimization. By doing so, we emphasize the strength of our method for the min-max routing problem.

2.1 Min-max routing vs. min-sum routing

The min-sum routing problems focus on minimizing the sum of total tour lengths, which includes resource-critical tasks such as the green vehicle routing problem [39]. The efficiency of the tour plays a crucial role in achieving this objective, as it directly impacts the sum of the tours [32]. Notably, the min-sum routing approach has shown promising performances in terms of NCO, demonstrating fast computation speeds and outperforming sophisticated heuristics in target tasks [40, 23, 25].

On the other hand, the min-max routing problem focuses on minimizing the maximum tour length among multiple agents, making it highly relevant for time-critical tasks such as disaster management and vaccine delivery. The objective here is to minimize the total completion time by leveraging all available resources. The min-max routing method takes into consideration the equity of tours among the multiple agents [32]. Recently, learning-based methods like the ScheduleNet [6], a reinforcement learning-based constructive solver, and Neuro-Cross Exchange (NCE) [35], a supervised learning-based improvement solver, have been proposed for addressing min-max routing problems. However, these methods still face challenges when dealing with large-scale problems.

2.2 Constructive solver vs. improvement solver

The NCO solver can be categorized into two kinds depending on its solution generation strategy; constructive solvers and improvement solvers. The constructive solvers aim to generate a solution from an empty set and sequentially add the solution components to complete the solution. An example of a constructive solver in the context of min-max routing is ScheduleNet [6]. ScheduleNet sequentially generates simultaneous actions of multiple agents by leveraging a graph neural network (GNN), which helps to capture the relationship between agents. The constructive solver is preferable in that it can satisfy the feasibility of the solution easily, by simply restricting decision space (e.g., masking scheme [10, 7, 18]) in the middle of the solution generation process. Our method is also a constructive solver for min-max routing, which first generates sequence auto-regressively and then reformulates generated sequence into simultaneous actions for the actual execution.

Improvement solvers [e.g., 17, 35, 41] revise completed solutions iteratively, which usually gives better performances than the constructive solver if the evaluation iteration is sufficiently provided. However, these methods usually suffer from low speed at the large scale problem. Also, making adequate improvement operations that satisfy the solution constraint requires domain knowledge, which is a downside compared to constructive solvers. For min-max routing, the NCE [35] is an improvement solver that gives powerful performances for moderate-sized problems but does not perform as effectively for large-scale problems.

3 Formulation of Min-max Routing as Sequential Generation

In our work, we focus on tackling min-max routing problems, which involve a scenario where a group of M agents needs to visit N cities with the objective of minimizing the maximum tour length among the individual tours of the agents. In this section, we present the formulation of min-max routing through the lens of sequential planning (see Fig. 1 for a detailed process).

Problem. The problem is the set of city locations (2D Euclidean) and initial locations of agents, which is represented as $\boldsymbol{x} = \{x_i\}_{i=1}^{N+M}$. $\{x_i\}_{i=1}^{N}$ is a location of cities to be visited and $\{x_i\}_{i=N+1}^{N+M}$ is an initial location of agents. Remarks that We can expand the definition of $\boldsymbol{x}$ so that it can include additional features required for other problems. For simplicity, we represent visit locations only.

Solution. The solution $\boldsymbol{e} = \{e_i\}_{i=1}^{N+M}$ is permutation index sequence of the problem: i.e., $e_i \in [1, N+M]$ and $e_i \neq e_j$ if $i \neq j$. Note that $e \in [1, N]$ stands for the cities, whereas $e \in$

$[N+1, N+M]$ stands for the dummy depot (i.e., the virtual return point) for each agent. The whole solution $\boldsymbol{e}$ can be partitioned M into sub-sequence, each of which represents the agent's tour, i.e., $\boldsymbol{e} = (e_1, \dots, e_{N+M}) = (\boldsymbol{e}^1, \dots, \boldsymbol{e}^M)$.

Agent Tour. The tour $\boldsymbol{e}^m = (e_1^m, ..., e_{L_m}^m)$ of agent m is sub-sequence with length L_m of a solution $\boldsymbol{e}$. Each tour must start at the depot, i.e. $e_{L_m}^m \in [N+1, N+M]$.

Cost. The cost is the maximum tour length among all agents' tours of $(\boldsymbol{e}^1, \dots, \boldsymbol{e}^M)$:

$$\mathcal{L}(\boldsymbol{e}^m; \boldsymbol{x}) := \sum_{i=2}^{L_m} ||x_{e_i^m} - x_{e_{i-1}^m}||_2 + ||x_{e_1^m} - x_{e_{L_m}^m}||_2$$
$$\mathcal{L}_{\text{cost}}(\boldsymbol{e}; \boldsymbol{x}) := \max \left\{ \mathcal{L}(\boldsymbol{e}^1; \boldsymbol{x}), ..., \mathcal{L}(\boldsymbol{e}^M; \boldsymbol{x}) \right\}.$$

Solver. The solver is the problem-conditional policy of generating sequence $\boldsymbol{e}$ of given problem condition $\boldsymbol{x}$ as follows:

$$\pi_\theta(\boldsymbol{e}|\boldsymbol{x}) = \prod_{i=1}^{N+M} \pi_\theta(e_i | e_1, ..., e_{i-1}, \boldsymbol{x}),$$

where θ is the deep neural network parameter of the policy π. To this end, the ultimate objective of our works can be expressed as finding an optimal parameter θ^* as follows:

$$\theta^* = \arg\min_\theta \mathbb{E}_{P(\boldsymbol{x})} \mathbb{E}_{\pi_\theta(\boldsymbol{e}|\boldsymbol{x})} \mathcal{L}_{\text{obj}}(\boldsymbol{e}; \boldsymbol{x}),$$

where $P(\boldsymbol{x})$ is the distribution of problem $\boldsymbol{x}$.

4 Architecture of Equity-Transformer

This section presents the architecture of Equity-Transformer $\pi_\theta(\boldsymbol{e}|\boldsymbol{x})$ which generates sequence $\boldsymbol{e}$ for given problem $\boldsymbol{x}$. See Fig. 2 high-level pipeline and see Appendix A for full architectures.

4.1 Encoder with multi-agent positional encoding

The encoder maps problem $\boldsymbol{x}$ into high dimensional representation H. Let $H = [h_1^\top, ..., h_{N+M}^\top] \in \mathbb{R}^{D \times (N+M)}$ be the D dimensional *encoded representation* of problem $X = [x_1^\top, ..., x_{N+M}^\top] \in \mathbb{R}^{2 \times (N+M)}$. The encoder $f_{\text{en}} : \mathbb{R}^{2 \times (N+M)} \to \mathbb{R}^{D \times (N+M)}$, a function mapping X to H is constructed as follows:

$$H = f_{\text{en}}(X) = f_{\text{en}}^{(L)} \circ \cdots \circ f_{\text{en}}^{(1)} \Big(\underbrace{f_{\text{PN}}\left(g_{\text{agent}}\left(X_{\text{agent}}\right)\right)}_{\text{Multi-agent Positional Encoding}} \oplus \underbrace{g_{\text{city}}\left(X_{\text{city}}\right)}_{\text{City Encoding}} \Big).$$

Where $f_{\text{en}}^{(l)}$ is the block function composed of multi-head attention, batch-normalization, and feed-forward neural network similar to that of [7, 38]. $f_{\text{PN}} : \mathbb{R}^{D \times M} \to \mathbb{R}^{D \times M}$ stands for the positional encoding on multi-agents, which is designed to impose order bias into multiple agents. Note this is a crucial technique for converting the simultaneous decision space into sequential space as it gives order bias between agents; see Fig. 5. $g_{\text{agent}} : \mathbb{R}^2 \to \mathbb{R}^D$ stands for the linear projection for M multi-agents of $X_{\text{agent}} = [x_{N+1}^\top, ..., x_{N+M}^\top] \in \mathbb{R}^{2 \times M}$. $g_{\text{city}} : \mathbb{R}^2 \to \mathbb{R}^D$ is linear projection (denoted as city encoding) for N cities of $X_{\text{city}} = [x_1^\top, ..., x_N^\top] \in \mathbb{R}^{2 \times N}$.

4.2 Context encoder for generating equity context

Our context encoder $f_{\text{ce}} : \mathbb{R}^{4D} \to \mathbb{R}^D$, which is a simple multi-layer perception (MLP), produces equity context $\boldsymbol{c}_t \in \mathbb{R}^D$ to be used for decoding (i.e., generating) multiple balanced tours. First, we design four contexts at the current step t: (1) problem context $\boldsymbol{h}_{\text{problem}} \in \mathbb{R}^D$, (2) agent context $\boldsymbol{h}_{\text{agent}} \in \mathbb{R}^D$, (3) scale context $\boldsymbol{h}_{\text{scale}} \in \mathbb{R}^D$, and (4) distance context $\boldsymbol{h}_{\text{distance}} \in \mathbb{R}^D$. These four contexts form a valuable source of task-equity information for multi-agents, as they compel agents to plan their tours while taking into account the remaining tasks of other agents. Then, we aggregate every above context and produce $\boldsymbol{c}_t$ via f_{ce}, i.e.,

$$\boldsymbol{c}_t = f_{\text{ce}}(\boldsymbol{h}_{\text{problem}} \oplus \boldsymbol{h}_{\text{agent}} \oplus \boldsymbol{h}_{\text{scale}} \oplus \boldsymbol{h}_{\text{distance}}).$$

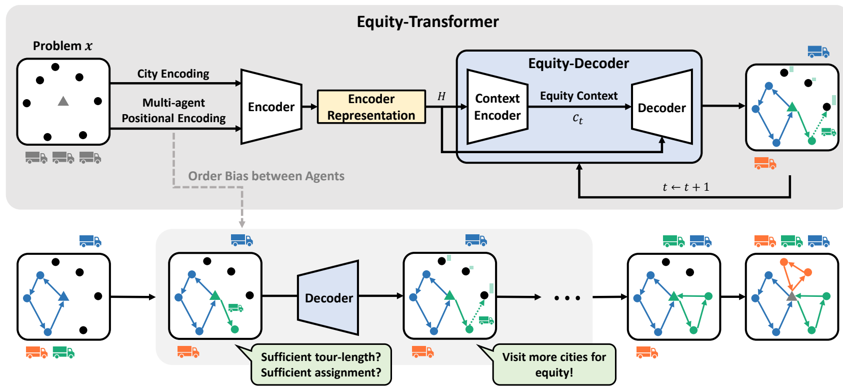


Figure 2: Illustration of Equity-Transformer, which is designed on the top of the attention model (AM) [7]. The multi-agent positional encoding introduces an order bias, represented by a virtual order (blue $\rightarrow$ green $\rightarrow$ orange), facilitating sequential planning for simultaneous decisions among multiple agents. The context encoder provides equity context, assisting the decoder in determining whether the current agent should continue visiting cities or return to the depot while considering the fairness of tour length among multiple agents.

Below, we offer a detailed design of each context.

Problem context. The problem context collects average representations $\boldsymbol{h}_{\text{problem}} = \frac{1}{M+N}\sum_{i=1}^{M+N} \boldsymbol{h}_i$. This aligns with the context embedding of the attention model (AM) [7], which is primarily intended to capture the global context of problem $\boldsymbol{x}$ by averaging each city and agent representation.

Agent context. We collect the representation of the currently active agent as follows: $\boldsymbol{h}_{\text{acting}} = \boldsymbol{h}_{i_t}$, where i_t denotes the index of the active agent at step t, i.e., $i_t \in [N+1, N+M]$. Additionally, we gather the representation of the city which the active agent is currently visiting: $\boldsymbol{h}_t$, where $t \in [1, N]$. Finally, we generate $\boldsymbol{h}_{\text{agent}} = g(\boldsymbol{h}_{\text{acting}} \oplus \boldsymbol{h}_t)$, where $g : \mathbb{R}^{2D} \rightarrow \mathbb{R}^D$ represents a linear projection. This context highlights the currently active agents.

Scale context. We incorporate the ratio between N_t and M_t as a scaling context, where N_t represents the number of remaining cities and M_t denotes the current number of idle agents at the depot, i.e., $N_t/M_t \in \mathbb{R}$. We then generate $\boldsymbol{h}_{\text{scale}} = g\left(N_t/M_t\right)$, where $g : \mathbb{R} \rightarrow \mathbb{R}^D$ represents a linear projection. This context offers valuable insights into the approximate number of cities an agent should visit to achieve equity. Consequently, the scale ratio can provide an inductive bias to the agent, aiding them in making informed decisions regarding whether to continue visiting additional cities or return to the depot.

Distance context. We utilize the distance information of the current state t. Firstly, we employ d^t_{source}, which represents the current tour length of the active agent from the depot. Secondly, we utilize d^t_{target}, which denotes the maximum distance between the depot and the remaining unvisited cities. Subsequently, we form $\boldsymbol{h}_{\text{distance}} = g(d^t_{\text{source}} \oplus d^t_{\text{target}})$, where $g : \mathbb{R}^2 \rightarrow \mathbb{R}^D$ is a linear projection. This information holds significant importance in terms of the equity of tour length among agents in the min-max routing problem. The context prompts the decoders to consider the agent's current tour length and remaining tasks, aiding in the decision-making process of whether to stop visiting (i.e., return to the depot) or continue the tour while considering the min-max tour length.

4.3 Decoder with equity context

The decoder f_{de} auto-regressively generates probability $\pi(e_t|\boldsymbol{c}_t, H) = f_{\text{de}}(\boldsymbol{c}_t, H)$ at step t, given *equity context* $\boldsymbol{c}_t \in \mathbb{R}^D$, and *encoder representation* H. First, multi-head attention between $\boldsymbol{c}_t$, and projected key $K_H \in \mathbb{R}^{(N+M)\times D}$ and value $V_H \in \mathbb{R}^{(N+M)\times D}$ from H produces $C \in \mathbb{R}^{(N+M)\times D}$, i.e.,

$$C = f_{\text{MHA}}\left(\boldsymbol{c}_t, K_H, V_H\right).$$

Then the probability $\pi(e_t|\boldsymbol{c}_t, H)$ is calculated by cross attention between $\boldsymbol{c}_t$ and projected key K_C of C as follows:

$$\pi(e_t|\boldsymbol{c}_t, H) = \text{SoftMax}\left(K_C \boldsymbol{c}_t^\top\right),$$

where the SoftMax is a function to generate probability from the vector of $\mathbb{R}^{M+N}$ so that the π can sample $e_t \in [M+N]$. Note we perform clipping and masking (to satisfy the constraint of the target problem) on the compatibility $K_C \boldsymbol{c}_t^\top$, see Appendix A for detail. This decoder has a similar structure of [7] where the most different part is on utilizing *equity context* $\boldsymbol{c}_t$ from the context encoder.

5 Training Scheme

This section provides the training scheme of Equity-Transformer in terms of pretraining and distributional adaptation. Let $\boldsymbol{\theta} = \{\theta_{\text{en}}, \theta_{\text{de}}, \theta_{\text{context}}\}$ be the parameter set of Equity-Transformer which contains encoder parameters θ_{en}, decoder parameters θ_{de}, and context coder parameters θ_{context}.

Training. The Equity-Transformer is trained with REINFORCE [42] with the shared baseline scheme [18, 25], which makes symmetric exploration for the combinatorial solution space.

Here is the REINFORCE training loss with the symmetric shared baselines:

$$\mathcal{L}_{\text{train}}(\boldsymbol{\theta}) = \mathbb{E}_{P(\boldsymbol{x})}\mathbb{E}_{\pi_\theta(\boldsymbol{e}|\boldsymbol{x})}\mathcal{L}_{\text{cost}}(\boldsymbol{e};\boldsymbol{x})$$

$$\nabla\mathcal{L}_{\text{train}}(\boldsymbol{\theta}) \approx \sum_{i=1}^{B}\sum_{j=1}^{L}\left(\mathcal{L}_{\text{cost}}(\boldsymbol{e}^{(i,j)};\boldsymbol{x}^{(i)}) - b_{\text{shared}}(\boldsymbol{e}^{(i,1)},...,\boldsymbol{e}^{(i,L)})\right).$$

Each $\boldsymbol{e}^{(i,1)},...,\boldsymbol{e}^{(i,L)}$ are sampled sequences from training solver given L symmetric $\boldsymbol{x}^{(i)}$: $p_\theta(\boldsymbol{e}|\mathcal{T}_1(\boldsymbol{x}^{(i)})),...,p_\theta(\boldsymbol{e}|\mathcal{T}_L(\boldsymbol{x}^{(i)}))$, where $\mathcal{T}_1,...,\mathcal{T}_L$ are symmetric transformation of problem instance $\boldsymbol{x}^{(i)}$. The training instances of $\boldsymbol{x}^{(1)},...,\boldsymbol{x}^{(B)}$ are sampled from $P(\boldsymbol{x})$; we use uniform distribution following [7]. The b_{shared} is shared baseline which is average value of L_{cost} among $\boldsymbol{e}^{(i,1)},...,\boldsymbol{e}^{(i,L)}$: $\frac{1}{L}\sum_{j=1}^{L}\mathcal{L}_{\text{cost}}(\boldsymbol{e}^{(i,j)};\boldsymbol{x}^{(i)})$. See [25] for a detailed training scheme.

Contextual finetuning. The distributional generalization capability of pretrained Equity-Transformer is remarkably strong, even when trained on a fixed small-scale problem of $N = 50$. However, to further improve its performance on larger-scale problems, we can efficiently adapt the model by finetuning only a relatively small number of parameters in $\boldsymbol{\theta}$, specifically θ_{context}.

6 Experiments

In this section, we present the experimental results of the Equity-Transformer model on two min-max routing problems: the multiple min-max traveling salesman problem (min-max mTSP) and the multiple pick-up and delivery problem (min-max mPDP).

Speed evaluation. All experiments were performed using a single NVIDIA A100 GPU and an AMD EPYC 7542 32-core processor as the CPU. Comparing the speed performance of classical algorithms (CPU-oriented) and learning algorithms (GPU-oriented) poses a significant challenge [7, 24], given the need for a fair evaluation. While certain approaches exploit the parallelizability of learning algorithms on GPUs, enabling faster solutions to multiple problems than classical algorithms [7], our method follows a serial approach in line with the prior min-max learning methods [6, 35]. It should be noted that when we leverage the parallelizability of our method, our approach can achieve speeds more than $100\times$ faster; refer to Appendix D for further details.

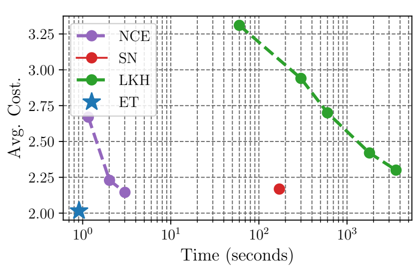
Training setting. For the training hyperparameters we set exactly the same hyperparameters for every task and experiment; see Appendix B for detail. We train Equity-Transformer on $N = 50$, and finetune it to target distribution of $N = 200, 500$. The training time for the min-max mTSP is approximately one day, while for the min-max mPDP, it takes around four days.

Experiments metric. It is important to carefully measure the performance comparison between methods, as there is often a trade-off between run time and solution quality. To this end, we present time-performance multi-objective graphs to compare tradeoffs between performance and computation time. In the result tables, we present the average cost achieved within a specific time limit, recognizing that every method has the potential to reach optimality given an unlimited amount of time.

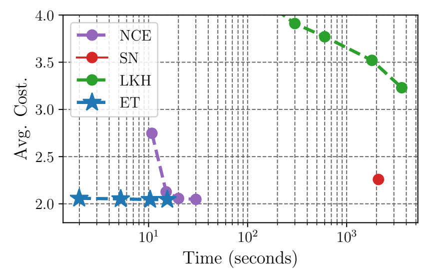
Target problem instances. To evaluate the performance of our methods, we report results on randomly generated synthetic instances of min-max mTSP and min-max mPDP at different problem scales of N and M. We generate the 100 problems set with a uniform distribution of node locations per scale. We conduct experiments with $N = 200, 500, 1000, 2000, 5000$ and set M such that $10 \leq N/M \leq 20$ by referring practical setting of min-max routing application [43]. To evaluate the

Table 1: Performance evaluation results on min-max mTSP. Every performance is average performance among 100 instances. The bold symbol indicates best performance. Average running times (in seconds) are provided in brackets. The notation 'LKH3 (seconds)' refers to the LKH3 method with a specified time limit, while 'OR-Tools (seconds)' indicates the OR-Tools implementation also with a defined time constraint.

		Classic-based				Learning-based		
N	M	LKH3 (60)	LKH3 (600)	OR-Tools (60)	OR-Tools (600)	SN [6]	NCE [35]	ET (*ours*)
200	10	2.52 (60)	2.08 (600)	4.97 (60)	2.22 (600)	2.35 (9.70)	2.07 (5.07)	**2.05** (0.36)
	15	2.39 (60)	2.03 (600)	4.82 (60)	2.15 (600)	2.13 (10.52)	1.97 (5.07)	**1.97** (0.37)
	20	2.29 (60)	2.02 (600)	3.74 (60)	2.04 (600)	2.07 (11.40)	1.96 (5.07)	**1.96** (0.37)
500	30	3.31 (60)	2.70 (600)	7.90 (60)	6.44 (600)	2.16 (171)	2.07 (5.20)	**2.02** (0.87)
	40	3.10 (60)	2.55 (600)	7.46 (60)	6.69 (600)	2.12 (276)	2.01 (5.38)	**2.01** (0.90)
	50	2.93 (60)	2.48 (600)	8.50 (60)	7.26 (600)	2.09 (217)	2.01 (5.05)	**2.01** (0.92)
1000	50	4.45 (60)	3.77 (600)	11.65 (60)	9.89 (600)	2.26 (2094)	2.13 (15.05)	**2.06** (1.72)
	75	3.71 (60)	3.26 (600)	13.16 (60)	11.50 (600)	2.17 (1678)	2.07 (15.05)	**2.05** (1.80)
	100	3.23 (60)	2.92 (600)	10.79 (60)	8.93 (600)	2.16 (1588)	2.05 (15.01)	**2.05** (1.79)
2000	100	6.60 (60)	4.61 (600)	20.99 (60)	18.85 (600)	OB	2.85 (43.96)	**2.09** (3.49)
	150	5.08 (60)	4.02 (600)	14.00 (60)	13.17 (600)	OB	2.83 (44.77)	**2.08** (3.41)
	200	4.13 (60)	3.36 (600)	11.00 (60)	10.41 (600)	OB	2.08 (30.30)	**2.08** (3.60)
5000	300	12.30 (60)	7.87 (600)	17.00 (60)	17.00 (60)	OB	2.97 (290)	**2.40** (8.78)
	400	8.85 (60)	6.15 (600)	13.00 (60)	13.00 (600)	OB	2.92 (204)	**2.21** (8.61)
	500	7.14 (60)	5.37 (600)	11.00 (60)	11.00 (600)	OB	2.89 (198)	**2.19** (9.02)



(a) mTSP with $(500, 30)$



(b) mTSP with $(1000, 50)$

Figure 3: Time-performance trade-off graph for mTSP. The left and bottom indicate the Pareto frontier.

performance on real-world datasets, we also report results on min-max mTSP benchmark [44] in Appendix C.

6.1 Performance evaluation on mTSP

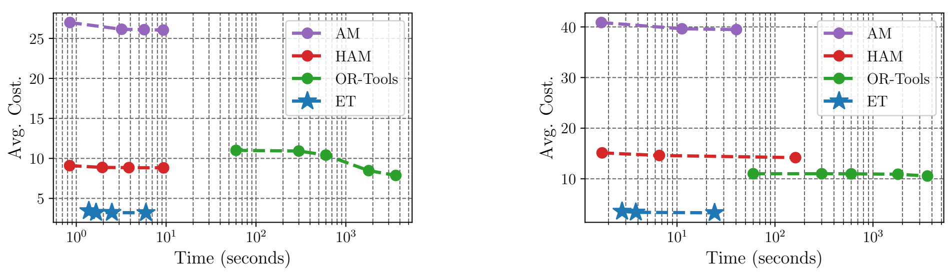
Baselines for mTSP. We consider two representative deep learning-based baseline algorithms: the ScheduleNet (SN) and Neuro Cross Exchange (NCE) for min-max mTSP. We have also included two classical heuristic methods, namely LKH3 [4] and OR-Tools [5], with respective time limits of 60 seconds and 600 seconds per instance. Specifically, LKH3 utilizes λ-opt improvement iterations to enhance the solution within the given time budget. The time limit directly influences the number of iterations performed (following the approach in [45]). Similarly, OR-Tools incorporates an iterative local search procedure for solution improvement, with the time limit governing the iterations of the local search following the approach in [7].

Results. The results in Table 1 demonstrate that the Equity-Transformer (denoted as 'ET' in tables and figures) outperforms all baselines with impressive speed. As the problem scale increases, the performance gap between Equity-Transformer and other methods widens further. Specifically, for $N = 1000$ and $M = 100$, ours achieves a cost of 2.05, significantly better than LKH3 (2.92) and NCE (2.16). Moreover, our method is $15.01/1.79 \approx 8.39\times$ faster than NCE and about $600/1.79 \approx 335\times$ faster than LKH3. The time-performance trade-off analysis shown in Fig. 3 confirms that our method outperforms every baseline and provides the Pareto frontier on multi-objective of time and cost.

For the large-scale problem of $N = 5000$, the SN method is notably slow, failing to produce a solution within 10,000 seconds per problem, making it out-of-budget (OB). While LKH3 and OR-Tools methods can provide solutions within the allotted time, their performance is inadequate due to the inherent difficulty of large-scale problems, requiring a significantly higher number of improvement

Table 2: Performance evaluation results on min-max mPDP. Every performance is average performance among 100 instances. The bold symbol indicates the best performance. Average running times (in seconds) are provided in brackets. The notation 'LKH3 (seconds)' refers to the LKH3 method with a specified time limit, while 'OR-Tools (seconds)' indicates the OR-Tools implementation also with a defined time constraint.

		Classic-based		Learning-based					
N	M	OR-Tools (60)	OR-Tools (600)	AM [7]	AM† [7]	HAM [46]	HAM† [46]	ET (*ours*)	ET† (*ours*)
200	10	20.96 (60)	18.76 (600)	15.88 (0.33)	15.65 (0.67)	5.69 (0.33)	5.30 (0.55)	5.03 (0.54)	**4.68** (0.55)
	15	13.96 (60)	8.46 (600)	15.88 (0.33)	15.57 (0.69)	5.21 (0.34)	5.09 (0.57)	3.91 (0.55)	**3.65** (0.56)
	20	10.67 (60)	5.70 (600)	15.88 (0.35)	15.55 (0.71)	5.21 (0.35)	5.09 (0.61)	3.39 (0.56)	**3.18** (0.61)
500	30	16.99 (60)	16.99 (600)	26.98 (0.82)	26.15 (3.10)	9.10 (0.84)	8.86 (1.92)	4.38 (1.33)	**4.11** (1.55)
	40	12.99 (60)	12.65 (600)	26.98 (0.86)	26.14 (3.17)	9.10 (0.84)	8.87 (1.95)	3.75 (1.36)	**3.52** (1.62)
	50	10.99 (60)	10.41 (600)	26.98 (0.85)	26.14 (3.20)	9.10 (0.88)	8.87 (1.95)	3.44 (1.38)	**3.23** (1.66)
1000	50	21.00 (60)	21.00 (600)	40.86 (1.61)	39.63 (11.28)	15.12 (1.63)	14.58 (6.35)	4.91 (2.63)	**4.73** (3.56)
	75	14.00 (60)	14.00 (600)	40.86 (1.68)	39.61 (11.44)	15.12 (1.70)	14.59 (6.41)	3.96 (2.65)	**3.77** (3.63)
	100	11.00 (60)	10.98 (600)	40.86 (1.69)	39.63 (11.24)	15.12 (1.72)	14.61 (6.61)	3.56 (2.75)	**3.38** (3.80)
2000	100	21.00 (60)	21.00 (600)	62.85 (3.24)	61.31 (24.98)	25.68 (3.40)	25.06 (15.17)	5.15 (5.22)	**4.91** (9.22)
	150	14.00 (60)	14.00 (600)	62.85 (3.34)	61.28 (25.43)	25.68 (3.40)	25.04 (16.26)	4.17 (5.31)	**3.97** (9.50)
	200	11.00 (60)	11.00 (600)	62.85 (3.35)	61.33 (26.33)	25.68 (3.50)	25.06 (16.47)	3.79 (5.43)	**3.62** (10.01)
5000	300	17.00 (60)	17.00 (600)	114.73 (8.30)	112.84 (180)	54.07 (34.65)	53.46 (279)	4.81 (52.66)	**4.60** (79.23)
	400	13.00 (60)	13.00 (600)	114.73 (8.31)	112.90 (182)	54.07 (34.44)	53.43 (283)	4.33 (54.86)	**4.11** (82.59)
	500	11.00 (60)	11.00 (600)	114.73 (8.33)	112.83 (186)	54.07 (34.46)	53.45 (286)	4.12 (54.77)	**3.88** (82.87)



(a) mPDP with (500, 50)

(b) mPDP with (1000, 100)

Figure 4: Time-performance trade-off graph for mPDP. The left and bottom indicate the Pareto frontier.

iterations for low-cost solutions. On the other hand, the NCE method surpasses classical approaches, as claimed in their main paper, but the Equity-Transformer outperforms NCE by a substantial margin of approximately $290/8.78 \approx 33\times$ faster speed and $(2.97 - 2.40)/2.97 \approx 19\%$ reduced cost.

6.2 Performance evaluation on mPDP

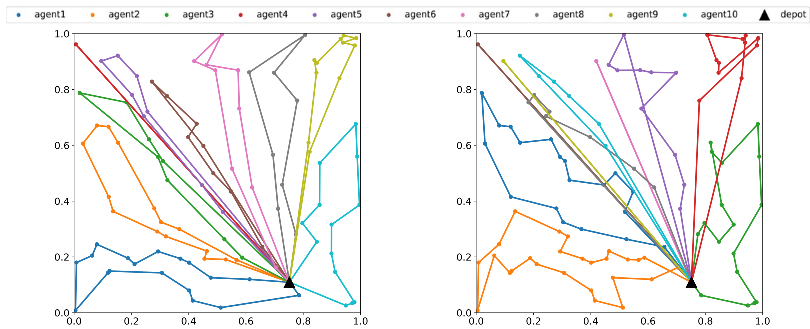
Baselines for min-max mPDP. We consider representative two deep learning-based baseline algorithms: AM [7] and heterogeneous AM (denoted as HAM) [46]. We retrain AM, and HAM using min-max objective; see Appendix B for detailed implementation. We also marked † by giving more trials for inference solutions such as sampling width [7] and augmentation width [18]; see Appendix B for detailed setting. We also include a classical heuristic method of OR-Tools [5], while LKH3 [4] cannot handle min-max mPDP to the best of our knowledge. We exclude the multi-agent PDP (MAPDP) model [47] due to the inaccessible source code.

The results presented in Table 2 demonstrate that our methods (i.e., ET and ET†) outperform all other baselines, aligning with the findings from the mTSP experiments. Compared to OR-Tools, ET† exhibits a remarkable speed improvement of $600/0.55 \approx 1901\times$, while reducing the objective cost by approximately $(18.76 - 4.68)/18.76 \approx 75\%$ at $N = 200$, $M = 10$. Moreover, Fig. 4 provides a visual representation of the outcomes, illustrating that despite granting the AM and HAM models more time to explore better solutions through sampling, Equity-Transformer consistently presents the Pareto frontier compared to other baselines.

Importantly, in certain instances, both AM and HAM produce identical cost values as the number of agents M increases. For instance, when $N = 500$, AM and HAM yield the same scores for $M = 30, 40, 50$. These methods were primarily designed to address min-sum problems (with HAM

Table 3: Ablation study for the combination of Equity-Transformer components. The MPE stands for multi-agent positional encoding, CE stands for contextual encoding, and CF stands for contextual finetuning. The $\emptyset$ stands for the plain AM [7] without our additional components. The average performance on min-max mTSP with 100 random instances for each (N, M) is reported.

Components	$(100, 5)$	$(100, 10)$	$(100, 15)$	$(200, 10)$	$(200, 15)$	$(200, 20)$
$\emptyset$	2.86	2.12	2.12	2.92	2.90	2.90
{MPE}	2.35	1.97	1.96	2.51	2.33	2.80
{CE}	2.52	1.97	1.95	2.28	2.01	1.98
{MPE, CE}	2.35	1.96	1.95	2.15	1.99	1.98
{MPE, CE, CF}	**2.29**	**1.95**	**1.95**	**2.06**	**1.97**	**1.96**



(a) {MPE} on mTSP $(100, 10)$ (b) $\emptyset$ on $(100, 10)$

Figure 5: Ablation study for the multi-agent positional encoding (MPE). (a) With MPE ($\{MPE\}$), the sequential order of agent1 $\rightarrow$ agent2 $\rightarrow$ agent3 $\rightarrow \cdots \rightarrow$ agent10 results in clock-wise tours. However, (b) without MPE ($\emptyset$), the sequential order is not correlated with the generation of tours.

especially focusing on min-sum mPDP), exhibiting a limited emphasis on leveraging the concept of *equity* among agents. These findings serve as compelling evidence supporting the success of our design choice centered around equity considerations.

6.3 Ablation study

To verify the impact of each component of Equity-Transformer on performance improvement, we conducted an ablation study. As shown in Table 3, all three components of Equity-Transformer resulted in clear performance gains across the six scaled problems. The $\emptyset$ represents the AM [7] without our components. This finding suggests that our novel Transformer design for min-max routing problems significantly contributes to the performance improvement observed in our approach.

Our results demonstrate the importance of both multi-agent positional encoding (MPE) at the encoder and the context encoder (CE) for supporting the decoder. With respect to MPE, we posit that incorporating an order bias into idle agents at the encoding stage enables the modeling of simultaneous decision-making as a sequential process. As shown in Fig. 5, the MPE contributes to creating an order bias among agents by generating cyclic sub-tours in the Euclidean space with specific orders. This can be interpreted as successful modeling of tour generation in the Euclidean space from multiple agents in the sequence space, which is the primary objective of MPE.

As for CE, we find it particularly useful in addressing distributional shift, which occurs when the distribution of the problem size is different from the training data; note that Equity-Transformer is trained on $N = 50$. This is especially evident in larger-scale problems, such as $N = 200$ in Table 3, where the purpose of CE is fully realized. Additionally, we have found that our context fine-tuning (CF) process proves to be generally beneficial for mitigating scale shifts.

7 Conclusion

In this paper, we introduced Equity-Transformer, a novel deep learning approach for multi-agent combinatorial optimization tasks. Our method surpassed existing state-of-the-art techniques and achieved a Pareto frontier concerning the trade-off between cost and runtime, both on two representative min-max routing tasks, mTSP and mPDP. We envision wide-ranging applications of Equity-Transformer in real-world services, particularly in multiple pick-up and delivery (mPDP) scenarios. Through our experiments, we have demonstrated the superior performance of Equity-Transformer, positioning it as a valuable tool for various delivery service industries aiming to enhance efficiency and reduce costs.

Acknowledgement

This work was supported by a grant of the KAIST-KT joint research project through AI2XL Laboratory, Institute of convergence Technology, funded by KT [Project No. G01210696, Development of Multi-Agent Reinforcement Learning Algorithm for Efficient Operation of Complex Distributed Systems].

References

[1] Christos H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237 – 244, 1977.

[2] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[3] Vašek Chvátal David Applegate, Robert Bixby and William Cook. Concorde tsp solver, 2023.

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

[5] Laurent Perron and Vincent Furnon. OR-Tools, 2019.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the TSP by policy gradient. In Willem-Jan van Hoeve, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 170–181, Cham, 2018. Springer International Publishing.

[13] 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.

[14] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *CoRR*, abs/1911.09539, 2019.

[15] Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2019.

[16] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning, 2019.

[17] Paulo R d O da Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In Sinno Jialin Pan and Masashi Sugiyama, editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pages 465–480, Bangkok, Thailand, 18–20 Nov 2020. PMLR.

[18] 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.

[19] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3243–3250, 2020.

[20] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning improvement heuristics for solving routing problems, 2020.

[21] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of 35th AAAI Conference on Artificial Intelligence*, pages 12042–12049, 2021.

[22] Aigerim Bogyrbayeva Yoon, Hanbum Ko, Sungbin Lim, Hyokun Yun, Changhyun Kwon, et al. A deep reinforcement learning approach for solving the traveling salesman problem with drone. *arXiv preprint arXiv:2112.12545*, 2021.

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

[25] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. *arXiv preprint arXiv:2205.13209*, 2022.

[26] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Hongliang Guo, Yuejiao Gong, and Yeow Meng Chee. Efficient neural neighborhood search for pickup and delivery problems. *arXiv preprint arXiv:2204.11399*, 2022.

[27] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. *arXiv preprint arXiv:2210.04123*, 2022.

[28] Haeyeon Kim, Minsu Kim, Joungho Kim, and Jinkyoo Park. Collaborative distillation meta learning for simulation intensive hardware design. *arXiv preprint arXiv:2205.13225*, 2022.

[29] Jiwoo Son, Minsu Kim, Hyeonah Kim, and Jinkyoo Park. Meta-sage: Scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. *arXiv preprint arXiv:2306.02688*, 2023.

[30] Hyeonah Kim, Minsu Kim, Sungsoo Ahn, and Jinkyoo Park. Symmetric exploration in combinatorial optimization is free! *arXiv preprint arXiv:2306.01276*, 2023.

[31] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances, 2020.

[32] Paulo M França, Michel Gendreau, Gilbert Laporte, and Felipe M Müller. The m-traveling salesman problem with minmax objective. *Transportation Science*, 29(3):267–275, 1995.

[33] Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, 40:100369, 2021.

[34] Luca Bertazzi, Bruce Golden, and Xingyin Wang. Min–max vs. min–sum vehicle routing: A worst-case analysis. *European Journal of Operational Research*, 240(2):372–381, 2015.

[35] Minjun Kim, Junyoung Park, and Jinkyoo Park. Learning to CROSS exchange to solve min-max vehicle routing problems. In *The Eleventh International Conference on Learning Representations*, 2023.

[36] Yujiao Hu, Yuan Yao, and Wee Sun Lee. A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems*, 204:106244, 2020.

[37] Yuhong Cao, Zhanhong Sun, and Guillaume Sartoretti. DAN: Decentralized attention-based neural network for the minmax multiple traveling salesman problem. *arXiv preprint arXiv:2109.04205*, 2021.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. 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 5998–6008. Curran Associates, Inc., 2017.

[39] Sevgi Erdoğan and Elise Miller-Hooks. A green vehicle routing problem. *Transportation research part E: logistics and transportation review*, 48(1):100–114, 2012.

[40] Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34, 2021.

[41] 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, 2021.

[42] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[43] Yi Ma, Xiaotian Hao, Jianye Hao, Jiawen Lu, Xing Liu, Tong Xialiang, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. *Advances in Neural Information Processing Systems*, 34:23609–23620, 2021.

[44] Raluca Necula, Mihaela Breaban, and Madalina Raschip. Tackling the bi-criteria facet of multiple traveling salesman problem with ant colony systems. In *2015 IEEE 27th international conference on tools with artificial intelligence (ICTAI)*, pages 873–880. IEEE, 2015.

[45] Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34, 2021.

[46] Jingwen Li, Liang Xin, Zhiguang Cao, Andrew Lim, Wen Song, and Jie Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2306–2315, 2021.

[47] Zefang Zong, Meng Zheng, Yong Li, and Depeng Jin. Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9980–9988, 2022.

[48] Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

A Illustration of Equity-Transformer Architecture

Fig. 6 shows the overall architecture of Equity-Transformer. Our architecture has three main components: an encoder, a decoder, and a context encoder. Our encoder has a similar structure to the attention model (AM) [7], with the only difference being the inclusion of a multi-agent positional encoder, as mentioned in Section 4. In the decoder, we incorporate the equity context $\boldsymbol{c}_t$, generated by the context encoder, as an additional input. Apart from this inclusion, the structure remains the same as that of the AM model.

As shown in Fig. 7, the context encoder produces the equity context $\boldsymbol{c}_t$ by referring to the current partial solution at step t and the encoder representation H. The illustration provides a detailed depiction as described in Section 4.2.

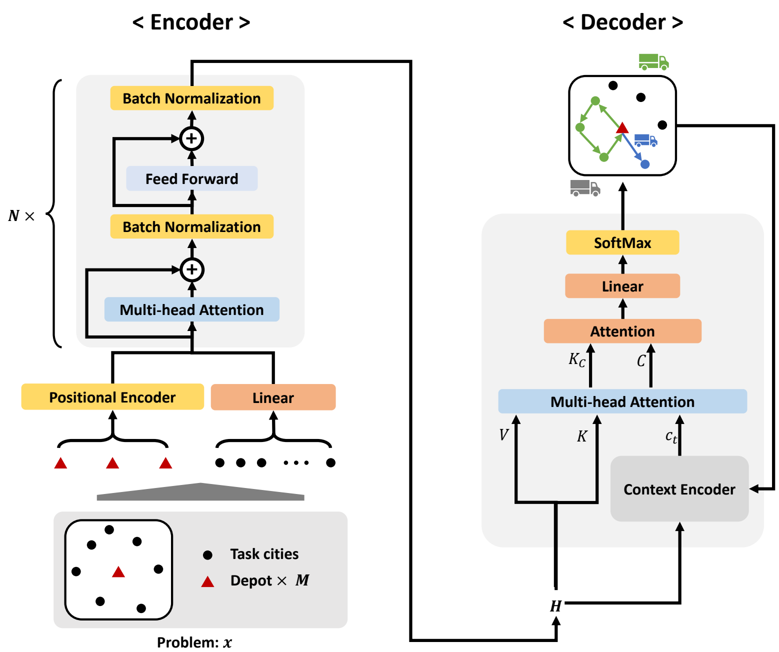


Figure 6: Illustration of Equity-Transformer. The N stands for the number of sequential layers, where we set $N = 3$.

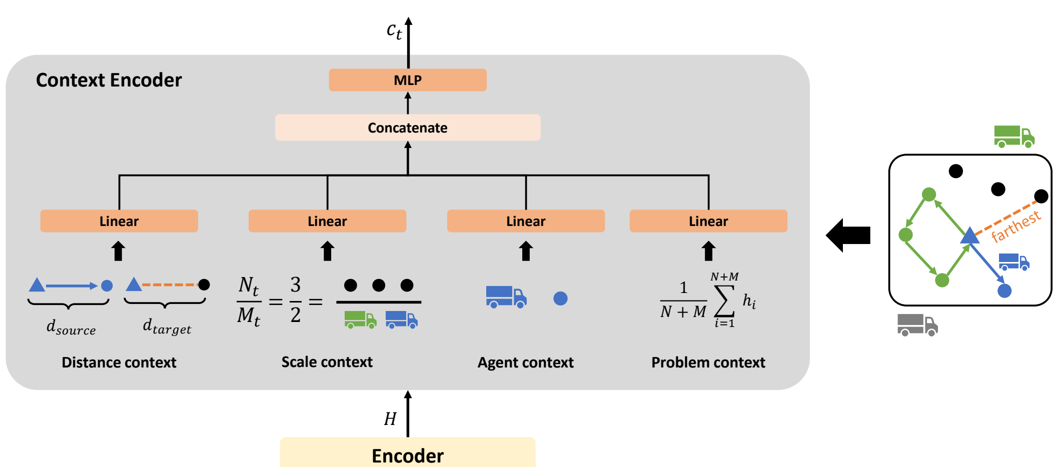


Figure 7: Illustration of context encoder.

B Detail of Experimental Setting

B.1 Datasets

In the mTSP experiment, we generate a random mTSP instance by randomly selecting N nodes from the unit square. As per convention, we designate the depot as the first index among the N nodes. Similarly, in the mPDP experiment, we generate the locations of the depot and customer nodes (pickup and delivery pairs) randomly and independently. Note that the first half of the customer nodes are assigned as pickup nodes, while the second half serves as delivery nodes.

B.2 Equity-Transformer

During the training step of Equity-Transformer (ET), we adopt the same hyperparameters as described in [7]. In the finetuning step, we specifically focus on adjusting the θ_{context} parameter with $N = 200, 500$ for mTSP and $N = 200$ for mPDP. For inference instances, we employ the finetuned model with $N = 200$ for mTSP and $N = 500$ for larger sizes and $N = 200$ for mPDP. Additionally, we incorporate an augmentation technique following the methodology outlined in [18].

Table 4: Hyperparameter setting for Equity-Transformer in mTSP and mPDP.

	Hyperparameters	mTSP	mPDP
Training	Learning rate	1e-4	1e-4
	Batch-size	512	512
	Epochs	100	100
	Epoch size	1,280,000	1,280,000
Finetuning	Learning rate	1e-5	1e-5
	Batch-size	128	128
	Finetuning-time	15h	15h
Inference	Augmentation	8	8

B.3 LKH3

LKH3 [4] is an extension of the Lin-Kernighan algorithm, an effective local-search heuristic for addressing TSP. LKH3 exhibits the capability to tackle not only TSP but also a diverse range of constrained routing problems, by minimizing a penalty function that quantifies the degree of a constraint violation. However, LKH3 does not support solving algorithms for the mPDP, limiting its application to the mTSP in our work. We use the executable program of LKH3, which is publicly available[2].

Table 5: Parameter setting for LKH3. It is worth noting that while we set specific values for the max trials and the runs, the algorithm predominantly terminated based on the time limit condition. For the remaining parameters not mentioned, we use the default values provided by the program.

Name	Value
Max trials	1000
Runs	1
Seed	3333
Time limit	{60, 300, 600, 1800, 3600}

B.4 OR-tools

OR-tools [5] is an open-source software that aims to solve various combinatorial optimization problems. OR-tools offers a wide range of solvers for tackling problems such as linear programming, mixed-integer programming, constraint programming, routing problems, and scheduling. Within the realm of routing problems, OR-tools can easily handle both mTSP and mPDP by simply incorporating additional constraints tailored to each problem. The installation guide for python OR-tools library

[2] http://webhotel4.ruc.dk/~keld/research/LKH-3

and example codes for various problems, including mTSP and mPDP, are readily available in their official website[3].

Table 6: Parameter setting for OR-tools. We use the same parameters for both mTSP and mPDP. Note that the global span cost coefficient should be sufficiently large to guarantee the desired objective for the min-max tasks. For the remaining parameters not mentioned, we use the default values suggested by OR-tools' official guides.

Name	Value
Global span cost coefficient	10000
First solution strategy	PATH_CHEAPEST_ARC
Local search metaheuristic	GUIDED_LOCAL_SEARCH
Time limit	{60, 300, 600, 1800, 3600}

B.5 ScheduleNet

ScheduleNet (SN) [6] is a method that utilizes a graph neural network (GNN) to sequentially generate simultaneous actions for multiple agents, effectively capturing the relationships between them. To conduct our experiment, we obtained the source code by contacting the author and followed the training procedure outlined in [6].

Table 7: Hyperparameter for training ScheduleNet in mTSP.

Name	Value
Learning rate	1e-4
Batch-size	512
Epochs	10,000
Epoch size	65,536
Discounting factor	0.9
Smoothing coefficient	0.1
Clipping parameter	0.2

B.6 Neuro Cross Exchange

The Neuro Cross Exchange (NCE) [35] is a supervised-learning-based improvement solver and a neural meta-heuristic technique that addresses vehicle routing problems (VRPs) by strategically swapping sub-tours among the vehicles to enhance solutions. To ensure consistency and comparability, we obtained the source code directly from the author by email and utilized a pre-trained model with the same parameters as described in [35]. And we restrict the time because the improvement method will get the optimal solution when it has enough time.

Table 8: Time limit for NCE in mTSP.

Number of cities (N)	Time limit
200	5.00
500	5.00
1,000	15.00
2,000	30.00
5,000	180.00

[3]https://developers.google.com/optimization

B.7 Attention Model

The Attention Model (AM) [7] serves as a fundamental component in numerous models, employing attention mechanisms to address a wide range of routing problems, including TSP, CVRP, PDP, and others, with remarkable effectiveness. During the training phase, we adhere to the recommended hyperparameters provided by the open-source code[4]. Additionally, we adopt a similar strategy for fine-tuning as ET.

Table 9: Hyperparameter setting for AM in mPDP.

	Name	Value
Training	Learning rate	1e-4
	Bats far and the ch-size	512
	Epochs	100
	Epoch size	1,280,000
Finetuning	Learning rate	1e-5
	Batch-size	128
	Finetuning-time	15h
Inference	Augmentation	8

B.8 Heterogeneous Attention Model

The Heterogeneous Attention Model (HAM) [46] is an adapted variation of the AM, specifically customized for addressing PDP. This model introduces additional attention mechanisms that are specifically designed to address the considerations of precedence constraint between the pickup node and the delivery node. During the training phase, we adhere to the recommended hyperparameters provided by the open-source code[5]. Additionally, we adopt a similar strategy for fine-tuning as ET.

Table 10: Hyperparameter setting for HAM in mPDP.

	Name	Value
Training	Learning rate	1e-4
	Batch-size	512
	Epochs	800
	Epoch size	1,280,000
Finetuning	Learning rate	1e-5
	Batch-size	128
	Finetuning-time	15h
Inference	Augmentation	8

[4] https://github.com/wouterkool/attention-learn-to-route

[5] https://github.com/Demon0312/Heterogeneous-Attentions-PDP-DRL

C Experimental Results on Real-world mTSP Dataset

This section presents the experimental results of a real-world mTSP dataset by converting the TSPLIB [48] into mTSP instances. Following the established convention [44] for converting TSPLIB to mTSP instances, we consider the TSPLIB instances as the locations of N cities, and we select the depot city as the first city in the list of cities. Additionally, we set the value of M as follows: $M = 10, 15, 20$ for problems with $200 < N < 500$, $M = 30, 40, 50$ for problems with $500 < N < 1000$, and $M = 50, 75, 100$ for problems with $N > 1000$. The results presented in Table 11 demonstrate that our ET† consistently outperforms all baselines across nearly all tasks.

Table 11: Performance evaluation results of real-world mTSP data converted from the TSPLIB [48] benchmark dataset. The running times (in seconds) are provided in brackets. ET† utilizes additional augmentation following the approach proposed in [25]. Specifically, 2000 augmentations are applied for cities with less than 500, 500 for cities with less than 1000, and 100 for cities with less than 2000. The "OB" represents "out-of-budget," indicating that the allocated budget for each instance is limited to 1 hour.

		Classic-based		Learning-based			
TSPLib	M	LKH3	OR-tools	SN [6]	NCE [35]	ET (*ours*)	ET† (*ours*)
kroA200	10	6417.19 (600)	**6223.22** (600)	8339.22 (11.72)	6281.18 (5.02)	6427.29 (0.35)	6294.89 (0.70)
	15	6417.19 (600)	**6223.22** (600)	6844.31 (10.87)	6280.73 (5.06)	**6223.22** (0.35)	**6223.22** (0.75)
	20	6418.51 (600)	**6223.22** (600)	7130.81 (9.88)	**6223.22** (5.02)	**6223.22** (0.36)	**6223.22** (0.72)
a280	10	648.89 (600)	944.42 (600)	715.95 (27.84)	643.59 (5.03)	646.89 (0.55)	**636.00** (1.25)
	15	668.89 (600)	951.98 (600)	648.98 (26.91)	617.67 (5.01)	611.80 (0.55)	**607.78** (1.35)
	20	656.11 (600)	943.60 (600)	649.96 (27.04)	**604.68** (5.03)	607.90 (0.58)	**604.68** (1.30)
lin318	10	10296.33 (600)	17546.77 (600)	10842.31 (29.35)	10042.13 (5.04)	10023.11 (0.70)	**9945.76** (1.58)
	15	10373.49 (600)	18406.43 (600)	9876.20 (31.07)	**9731.17** (5.07)	**9731.17** (0.69)	**9731.17** (1.60)
	20	10854.49 (600)	17628.86 (600)	9933.23 (34.55)	**9731.17** (5.12)	**9731.17** (0.64)	**9731.17** (1.64)
pr439	10	26206.85 (600)	58355.61 (600)	25807.61(105)	29685.56 (5.04)	25651.63 (0.71)	**24374.03** (2.37)
	15	22457.51 (600)	58355.50 (600)	22968.29 (102)	24689.55 (5.04)	22325.89 (0.78)	**21833.94** (2.42)
	20	24503.16 (600)	58355.61 (600)	22539.62.29 (103)	21828.70 (5.00)	22112.27 (0.76)	**21703.51** (2.45)
u574	30	8799.53 (600)	19391.32 (600)	6875.21 (244)	**6641.51** (5.02)	**6641.51** (0.95)	**6641.51** (1.62)
	40	8051.49 (600)	15923.89 (600)	6731.26 (257)	**6641.51** (5.08)	**6641.51** (1.03)	**6641.51** (1.64)
	50	7733.13 (600)	14191.82 (600)	6859.72 (289.65)	**6641.51** (5.16)	**6641.51** (1.01)	**6641.51** (1.71)
p654	30	13317.00 (600)	25551.51 (600)	14649.50 (453.36)	16069.91 (5.05)	15905.85 (1.10)	**12794.86** (1.92)
	40	13667.81 (600)	25547.37 (600)	14627.26 (393)	16246.51 (5.01)	13016.20 (1.13)	**12747.33** (2.05)
	50	13187.50 (600)	25547.37 (600)	14531.31 (494.67)	14832.95 (5.05)	12788.09 (1.14)	**12501.64** (2.05)
rat783	30	2217.40 (600)	5105.11 (600)	1380.92 (628)	1381.56 (6.26)	1319.97 (1.30)	**1271.52** (2.61)
	40	1872.14 (600)	5105.11 (600)	1352.61 (792)	1423.14 (6.31)	1254.60 (1.31)	**1237.64** (2.60)
	50	1639.60 (600)	4004.67 (600)	1323.87 (787)	1410.63 (6.38)	1249.93 (1.30)	**1231.69** (2.67)
pr1002	50	54569.48 (600)	159502.40 (600)	37844.23 (1647)	34894.64 (15.03)	**34365.02** (1.64)	34465.98 (2.02)
	75	50657.38 (600)	117172.89 (600)	36793.27 (1678)	**33861.63** (15.03)	34263.05 (1.70)	**33861.63** (2.06)
	100	42902.62 (600)	135228.92 (600)	35654.55 (1855)	**33861.63** (15.09)	**33861.63** (1.85)	**33861.63** (2.19)
pcb1173	50	8530.43 (600)	35366.75 (600)	6715.26 (2530)	6667.08 (15.01)	6623.27 (1.93)	**6607.14** (2.39)
	75	9844.05 (600)	25881.61 (600)	6769.86 (2622)	6539.95 (15.01)	6562.89 (1.98)	**6528.86** (2.51)
	100	8770.60 (600)	25237.96 (600)	6765.07 (2392)	**6528.86** (15.01)	**6528.86** (2.03)	**6528.86** (2.49)
d1291	50	11268.07 (600)	35976.42 (600)	11085.14 (2735)	11163.11 (16.81)	9955.28 (2.20)	**9858.99** (2.67)
	75	11455.59 (600)	35976.42 (600)	10776.30 (3242)	11003.23 (16.74)	**9858.99** (2.17)	**9858.99** (2.72)
	100	9998.52 (600)	30250.32 (600)	10419.44 (3511)	10904.31 (16.69)	**9858.99** (2.21)	**9858.99** (2.86)
rl1304	50	50916.48 (600)	179168.59 (600)	*OB*	45209.16 (17.48)	33617.21 (2.11)	**33072.96** (2.63)
	75	39774.59 (600)	178270.00 (600)	*OB*	42899.00 (17.77)	33009.00 (2.80)	**32473.34** (2.80)
	100	53956.65 (600)	118561.04 (600)	*OB*	41927.05 (17.57)	32557.78 (2.26)	**32134.47** (2.80)
u1432	50	39189.72 (600)	85050.22 (600)	*OB*	15282.77 (20.47)	13027.50 (2.31)	**12830.64** (2.95)
	75	24844.31 (600)	80893.17 (600)	*OB*	15007.52 (20.54)	12835.57 (2.43)	**12625.89** (3.08)
	100	22187.92 (600)	70304.41 (600)	*OB*	15241.95 (20.45)	12521.75 (2.47)	**12155.28** (3.10)
fl1577	50	5631.54 (600)	15232.65 (600)	*OB*	6586.44 (24.92)	4531.97 (2.69)	**4158.05** (3.41)
	75	9446.27 (600)	15232.65 (600)	*OB*	6356.05 (24.45)	4278.85 (2.62)	**4071.33** (3.44)
	100	7356.55 (600)	15232.65 (600)	*OB*	6089.94 (24.59)	**4069.46** (2.66)	4087.09 (3.62)
u1817	50	16776.31 (600)	37910.67 (600)	*OB*	7672.36 (32.66)	6549.83 (2.96)	**6490.56** (4.13)
	75	15685.26 (600)	37910.67 (600)	*OB*	7283.72 (32.84)	6540.25 (2.98)	**6424.96** (4.07)
	100	14071.51 (600)	28891.86 (600)	*OB*	7174.80 (32.67)	6431.92 (3.04)	**6413.51** (4.26)

D Speed Evaluation for Serial and Parallel Process

This section provides an evaluation of the speed performance of Equity-Transformer in two distinct scenarios: the serial process, where instances are solved one by one in a sequential manner, and the parallel process, where instances are solved concurrently. The results, illustrated in Appendix D, clearly demonstrate that Equity-Transformer exhibits remarkable parallelization capabilities, resulting in a speed improvement of approximately 153 times compared to the serial process for the mTSP with 500 instances and $N = 1000$. This significant enhancement underscores the potential of Equity-Transformer to efficiently handle multiple routing problem requests simultaneously in real-world situations. It is worth noting that even in the serial process, as shown in the main table, Equity-Transformer surpasses the baseline method in terms of speed, without leveraging any parallelization advantages.

Table 12: Comparison of total time for solving instances in mTSP with the serial process and parallel process in Equity-Transformer. The s stands for the seconds.

N	Number of instances	Serial Process	Parallel Process
200	10	3.30s	1.36s
	100	33.67s	1.51s
	500	169.55s	2.98s
500	10	8.37s	1.93s
	100	84.13s	2.84s
	500	449.05s	3.68s
1000	10	16.48s	3.02s
	100	163.75s	4.53s
	500	815.52s	5.35s