

Hierarchical Decomposition Framework for Steiner Tree Packing Problem

Hanbum Ko^{1,*}^a, Minu Kim^{2,*}^b, Han-Seul Jeong³^c, Sunghoon Hong³^d, Deunsol Yoon³^e,
Youngjoon Park³^f, Woohyung Lim³^g, Honglak Lee³^h, Moontae Lee³ⁱ, Kanghoon Lee³^j,
Sungbin Lim⁴^k and Sungryull Sohn³^l

¹Department of Artificial Intelligence, Korea University, Korea

²Kim Jaechul Graduate School of AI, KAIST, Korea

³LG AI Research, U.S.A.

⁴Department of Statistics, Korea University, Korea

{hanseul.jeong, sunghoon.hong, dsyoon,yj.park, w.lim}@lgresearch.ai,

Keywords: Hierarchical Reinforcement Learning, Neural Combinatorial Optimization, Steiner Tree Packing Problem.


Abstract: In this paper, we address the complex combinatorial optimization (CO) challenge of efficiently connecting objects at minimal cost, specifically within the context of the Steiner Tree Packing Problem (STPP). Traditional methods often involve subdividing the problem into multiple Steiner Tree Problems (STP) and solving them in parallel. However, this approach can fail to provide feasible solutions. To overcome this limitation, we introduce a novel hierarchical combinatorial optimizer (HCO) that applies an iterative process of dividing and solving sub-problems. HCO reduces the search space and boosts the chances of getting feasible solutions. This paper proposes for the first time a learning-based approaches to address STPP, introducing an iterative decomposition method, HCO. Our experiments demonstrate that HCO outperforms existing learning-based methods in terms of feasibility and the quality of solutions, and showing better training efficiency and generalization performance than previous learning-based methods.


1 INTRODUCTION


The challenge of optimizing connections at minimal cost is crucial across various domains. In logistics and transportation networks, efficient route design is crucial for connecting warehouses, markets, and facto-


ries. Urban planning requires the cost-effective linkage of infrastructure, while chip design demands the minimal-cost connection of electronic components to enhance data movement and signal stability. Determining the optimal configuration from numerous possibilities is a complex combinatorial optimization (CO) problem, often referred to as the Steiner Tree Packing Problem (STPP). STPP aims to identify a set of minimum-cost edges that connect specific terminal nodes without overlap, and it can be decomposed into simpler Steiner Tree Problems (STP) (Karp, 1972), where the objective is to connect all given terminal nodes with the least number of edges.


The approach of dividing the CO problem into multiple sub-problems and solving the divided sub-problems in parallel—before even solving the first sub-problem—has seen numerous attempts in various problems such as the vehicle routing problem (VRP) and the traveling salesman problem (TSP) (Nowak-Vila et al.; Hou et al.; Fu et al.; Ye et al.). This


^a <https://orcid.org/0000-0002-2601-0974>


^b <https://orcid.org/0000-0002-8126-4387>


^c <https://orcid.org/0000-0002-3525-8699>


^d <https://orcid.org/0000-0002-6745-5005>


^e <https://orcid.org/0000-0002-1069-112X>


^f <https://orcid.org/0000-0002-3950-3065>


^g <https://orcid.org/0000-0003-0525-9065>

^h <https://orcid.org/0000-0002-4109-327X>

ⁱ <https://orcid.org/0000-0001-5542-3463>

^j <https://orcid.org/0000-0002-2077-7146>

^k <https://orcid.org/0000-0003-2684-2022>

^l <https://orcid.org/0000-0001-7733-4293>

*This work was conducted during an internship at LG AI Research.

separation of concerns, which involves splitting the model into one that only divides the problem and another that solves the divided problems, enhances time efficiency by solving the problems in parallel. The STPP is another CO problem that can be divided into multiple sub-problems (i.e., STP). However, this method significantly increases the risk of not obtaining a feasible solution. To address this disadvantage, we propose a hierarchical combinatorial optimizer (HCO) that iteratively processes the dividing and solving stages. An example of this is shown in Figure 1. The Two-stage Divide Method (TAM) (Hou et al., 2022) involves dividing all sub-problems before solving any of them, whereas the HCO represents a case where the process of dividing and solving sub-problems is done iteratively. In the leftmost graph, which is an STPP instance, circular nodes represent normal nodes, while square nodes represent terminal nodes. The different colors of the square nodes indicate different types of terminal nodes, and the objective of STPP is to connect terminal nodes of the same type with minimal cost. The main issue with applying the TAM to solve the STPP is that it assumes all nodes of a specific color (apricot, in this case) are selected before choosing the next sub-problem to divide. This leads to a scenario where green nodes may not have the opportunity to connect with the only feasible node. Conversely, the HCO, by re-evaluating the solution of the divided STP problem, creates the possibility to connect the green terminal nodes.

We refer to problems where a feasible solution may not emerge as feasibility-hard problems. In this study, we propose the HCO method to more effectively address these challenges. Additionally, we mathematically demonstrate that HCO can reduce the search space through latent mapping to a smaller solution space. We argue that HCO has the following advantages: First, as observed in the examples in Figure 1, it can more effectively find feasible solutions. Second, by reducing the search space, it alleviates the burden on models that must consider both the objective and feasibility of the CO problem simultaneously. Third, it benefits from time efficiency by utilizing accurate and fast models that solve sub-problems well. Lastly, the model that divides a large problem into smaller ones reduces the distribution shift issue by partitioning the graph. This ensures that sub-problems only see problems of similar size STP, thus offering advantages in generalization.

In this paper, we detail the Markov decision problem (MDP) formulation for finding solutions to CO problems and introduce the application of this formulation to the STPP. We then describe how we apply this formulation to solve problems using a hierarchi-

cal policy involving high-level and low-level policies. Our experiments demonstrate that the HCO, defined in this manner, more successfully finds feasible solutions and produces higher quality solutions compared to the method of dividing and solving the problem at once (TAM) and the method of solving STPP at once without hierarchical decomposition (Khalil et al.; Kool et al.).

Our contributions are summarized as follows:

- To our knowledge, this is the first work to solve Steiner tree packing problem using an end-to-end learning framework.
- We propose a novel decomposition approach for general CO problems that results in sub-problems with smaller search spaces.
- We demonstrate that HCO is advantageous over other methods in terms of finding feasible solutions and the optimal gap of the found solutions. Additionally, HCO exhibits the ability to generalize to larger problem sizes.

2 RELATED WORKS

Combinatorial Optimization with Feasibility-Hard Constraints. There have been few attempts to directly tackle CO problems with feasibility-hard constraints using RL. Ma et al. (2021) proposed learning two separate RL models, with each model respectively solving the constraint satisfaction and objective optimization problems. Cappart et al. (2021) manually shaped the reward to bias the RL process toward predicting feasible solutions and combined this approach with constraint programming methods to guarantee the feasibility of the solutions. Our work indirectly tackles the feasibility-hard constraint by decomposing the given constraint satisfaction problem into two easier sub-problems with smaller problem size and search space, allowing the learning algorithm to efficiently solve each sub-problem.

Decomposition of Combinatorial Optimization Problem. Several decomposition methodologies have been introduced to address a variety of CO challenges. Nowak-Vila et al. (2018) introduced a Divide-and-Conquer (DnC) framework focusing on scale-invariant problems, which assumes the problem can be split into sub-problems, solved independently, and subsequently merged to form a full solution. Hou et al., Fu et al. and Ye et al. proposed the domain-specific approaches to divide problems and merge partial solutions (via heatmap and MCTS) for VRP

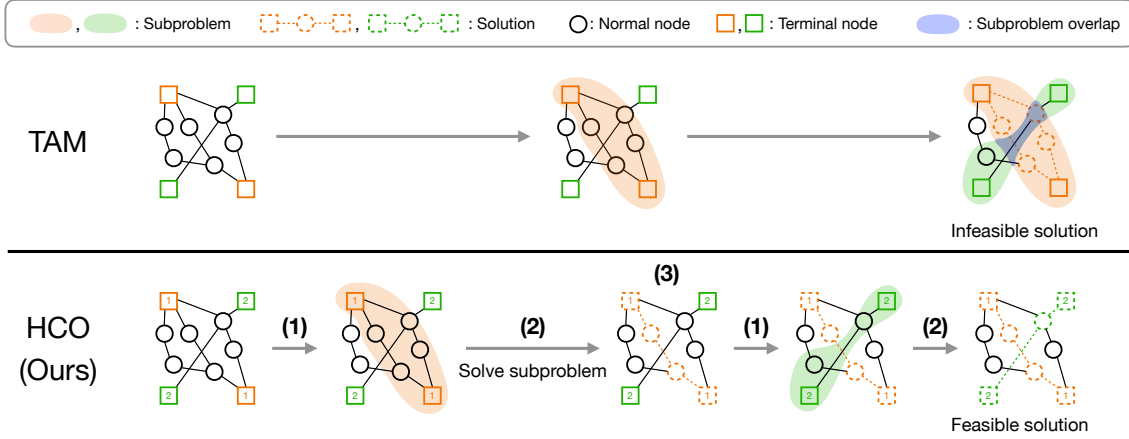


Figure 1: Comparison of two decomposition methods in the Steiner tree packing problem (STPP). (Top) TAM divides the problem into subproblems all at once, allowing overlapping vertices between them. The entire solution may be infeasible if the solutions of subproblems overlap. (Bottom) HCO, on the other hand, solves the problem iteratively by: 1) selecting a sub-graph to define a subproblem, 2) solving the identified subproblem, 3) reflecting its solution, and repeating this process until the entire problem is solved. Since HCO defines each subproblem by considering all previous solutions, it can more effectively find a feasible solution compared to TAM.

and TSP, respectively. Despite their utility, the applicability of such DnC-based methods is limited by their reliance on scale invariance, often leading to infeasible solutions when this assumption is violated. In a different vein, local search-based methods by Song et al. and Li et al. require an initial solution to iteratively refine decision variables, which is a notable constraint. Wang et al. (2021) further investigated a bi-level formulation, where the high-level problem modifies the given problem instance and the low-level problem solves the modified instance, to ease problem solving. However, this occasionally resulted in more complex instances (*e.g.*, a feasible solution may not exist) and suboptimal solutions. Our algorithm HCO also adopts the bi-level formulation but sidesteps these issues by preserving the original problem structure.

3 PRELIMINARIES

3.1 Combinatorial Optimization as Markov Decision Process

Combinatorial optimization (CO) is a mathematical optimization over a finite set, with a discrete feasible solution space. Formally, a combinatorial optimization problem can be written as follows.

$$\arg \min_{\mathbf{x} \in X} \{f(\mathbf{x}) : \mathbf{x} \in \mathcal{F}\} \quad (1)$$

where X is a finite support for the variable \mathbf{x} , $\mathcal{F} \subset X$ is a set of feasible solutions¹, and $f : X \rightarrow \mathbb{R}$ is an objective function of the CO problem. For instance, a mixed integer linear programming (MILP) problem with n variables and m constraints can be written in the form

$$\arg \min_{\mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n-p}} \{\mathbf{c}^\top \mathbf{x} : \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\} \quad (2)$$

where $\mathbf{A} \in \mathbb{R}^{n \times m}$, $\mathbf{b} \in \mathbb{R}^m$, and $\mathbf{c} \in \mathbb{R}^n$.

Most of the CO problems can be formulated as a Markov Decision Process (MDP) Khalil et al.; Gasse et al.. Formally, it makes two assumptions to the CO problem: 1) the solution space X of the original problem (1) is a finite vector space and 2) the objective f is *linear* on X , so that for any given decomposition of X into direct sum of subspaces $X = X_1 \oplus \dots \oplus X_n$, we have $f(\mathbf{x}) = \sum_{i=1}^n f(\mathbf{x}_i)$ for each $\mathbf{x}_i \in X_i$. Then, the original problem (1) can be written as the following sequential decision making problem:

$$\arg \min_{\substack{\mathbf{x}_t \in X_t, \forall t=1, \dots, H \\ X = X_1 \oplus \dots \oplus X_H}} \left\{ \sum_{t=1}^H f(\mathbf{x}_t) : \sum_{t=1}^H \mathbf{x}_t \in \mathcal{F} \right\}. \quad (3)$$

The sequential decision can be thought of choosing for each timestep t an action $\mathbf{x}_t \in X_t$, to receive a reward $\mathcal{R}(s_t, a_t) = -f(a_t)$ and a large negative penalty $c \leq -\sup_{\mathbf{x} \in X} f(\mathbf{x})$ if and only if any future choice of action inevitably leads to an infeasible solution at the end of the horizon. The optimal policy $\pi^* \in \Pi$ for the original problem can be found upon maximizing the expected return.

¹ \mathcal{F} is either discrete itself or can be reduced to a discrete set.

$$\pi^* = \arg \max_{\pi \in \Pi} \mathbb{E}^\pi \left[\sum_{t=1}^H \mathcal{R}(s_t, a_t) \mid s_0 \right]. \quad (4)$$

We defer the rest of the details to Appendix. Note that for some CO problems (e.g., TSP, MVC, Max-Cut), carefully designing the action space can make the constraint trivially satisfied (Khalil et al., 2017), where in this case, reinforcement learning algorithm can efficiently solve the problem. However, when the constraint satisfaction is not guaranteed, the reinforcement learning methods often suffer from the sparse reward problem, and does not learn efficiently. In this work, we focus on the challenging CO problems, where designing action space cannot guarantee the constraint satisfaction (i.e., *feasibility-hard* constraint): the Steiner tree packing problem.

3.2 Steiner Tree Packing Problem

A Steiner Tree Problem (STP) can be thought of as a generalization of a minimum spanning tree problem, where given a weighted graph and a subset of its vertices (called terminals), one aims to find a tree (called a Steiner tree) that spans all terminals (but not necessarily all nodes) with minimum weights. Although minimum spanning tree problem can be solved within polynomial time, the Steiner tree problem itself is a NP-complete combinatorial problem (Karp, 1972). Formally, let $G = (V, E)$ be an undirected weighted graph, w_e for $e \in E$ its edge weights, and $T \subset V$ be the terminals. Then, a Steiner tree \mathcal{S} is a tree that spans T such that its edge weight is minimal. Hence, the optimization problem for STP can be written as follows.

$$\arg \min_{\mathbf{x} \in 2^E} \left\{ \sum_{e \in \mathbf{x}} w_e : \mathbf{x} \in \Sigma_T \right\} \quad (5)$$

where 2^E is a power set of E , and Σ_T is a set of all Steiner trees that span T . A more generalized version of the above Steiner tree problem is called the Steiner Tree Packing Problem, (STPP) where one has a collection \mathcal{T} of N disjoint non-empty sets T_1, \dots, T_N of terminals called *nets*, that has to be *packed* with disjoint Steiner trees S_1, \dots, S_N spanning each of the nets T_1, \dots, T_N . The optimization problem for STPP can be written similarly, with N variables.

$$\arg \min_{\mathbf{x}_1, \dots, \mathbf{x}_N \in 2^E} \left\{ \sum_{\substack{n \leq N \\ e \in \mathbf{x}_n}} w_e : \mathbf{x}_n \in \Sigma_{T_n}, G[\mathbf{x}_n] \cap G[\mathbf{x}_m] = \emptyset \right\} \quad (6)$$

where $G[\mathbf{x}]$ is a subgraph of G generated by $\mathbf{x} \subset E$.

4 BI-LEVEL DECOMPOSITION FOR STEINER TREE PACKING PROBLEM

The goal of this section is to formulate our bi-level framework for STPP and the corresponding MDP, which allows us to use a hierarchical reinforcement learning policy that efficiently learns to solve CO problems.

Let us introduce a continuous surjective latent mapping $\phi : X \rightarrow Y$ onto a vector space Y such that $|Y| \ll |X|$. Then, the problem (1) admits a *hierarchical* solution concept:

$$\mathbf{y}^* = \arg \min_{\mathbf{y} \in Y} \{f(\mathbf{L}(\mathbf{y})) : \phi^{-1}(\mathbf{y}) \cap \mathcal{F} \neq \emptyset\} \quad (7)$$

$$\text{where, } \mathbf{L}(\mathbf{y}) := \arg \min_{\mathbf{x} \in \phi^{-1}(\mathbf{y})} \{f(\mathbf{x}) : \mathbf{x} \in \mathcal{F}\}. \quad (8)$$

We refer to problem (7) as a *high-level* problem, and (8) as a *low-level* sub-problem induced by the high-level action \mathbf{y} in (7). Note that the hierarchical solution concept still attains an optimality guarantee of the original problem, since ϕ is a surjection and X is finite.

The advantages of such hierarchical formulation are: (i) searching for feasible solutions over Y rather than X reduces the size of search space; (ii) learning to obtain an optimal solution can be done by two different learnable agents, (namely the high-level agent and the low-level agent for (7) and (8), respectively) where the task for each agent is reduced to be easier than the original problem, and (iii) the generalization capability (with respect to the problem size) increases when using learnable agents, since the high-level agent can be made to always provide sub-problems (for the low-level agent) with the same size, regardless of the size of the original problem.

4.1 Hierarchical Decomposition for Steiner Tree Packing Problem

For CO problems defined on a weighted graph $G = (V, E)$ with edge weights w_e for each $e \in E$, it is straight-forward and beneficial to choose a latent mapping $\phi : E \rightarrow V$ from the set of edges to the set of nodes.² Specifically, we consider a version of ϕ such that for any input $\mathbf{x} \subset E$, $u \in \phi(\mathbf{x})$ if and only if $(u, v) \in \mathbf{x}$ for some $v \in V(G)$. Such a mapping ϕ satisfies $\phi^{-1}(\mathbf{y}) = G[\mathbf{y}]$ for any set of vertices $\mathbf{y} \subset V$, where we slightly overload the notation for the gener-

²Since $|V| \approx O(\sqrt{|E|})$, so that $|Y| \ll |X|$ for large graphs.

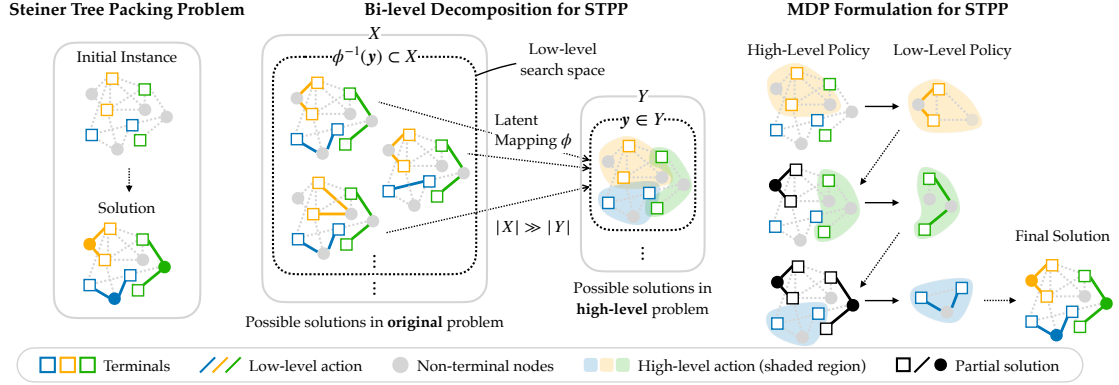


Figure 2: (Left) We tackle the Steiner tree packing problem, which aims to find a minimum-weight tree spanning all the terminal nodes (square boxes) for each type (color) without overlap. (Middle) We propose to decompose the given problem into high-level and low-level sub-problems via mapping $\phi: X \mapsto Y$, which are solved separately. This facilitates learning since the search spaces are much smaller for high-level problem $|Y| \ll |X|$ and low-level problem $|\phi^{-1}(y)| \ll |X|$ compared to the original problem. (Right) In MDP formulation, the high-level agent chooses a set of nodes (shaded region) to define a sub-graph for each terminal type (color), and low-level agent finds a tree spanning all the terminal nodes within the sub-graph.

ated subgraph $G[\mathbf{y}]$. For example, the high-level problem of a STPP (6) can be written as follows.

$$\arg \min_{\mathbf{y}_1, \dots, \mathbf{y}_N \in 2^V} \left\{ \sum_{\substack{n \leq N \\ e \in L(\mathbf{y}_n)}} w_e : L(\mathbf{y}_n) \in \Sigma_{T_n}, \mathbf{y}_n \cap \mathbf{y}_m = \emptyset \right\} \quad (9)$$

which is now a node-selection problem, (instead of the original edge selection problem) where $L(\mathbf{y}_n)$ is a solution of the low-level subproblem:

$$L(\mathbf{y}_n) := \arg \min_{\mathbf{x} \in E(G[\mathbf{y}_n])} \left\{ \sum_{e \in \mathbf{x}} w_e : \mathbf{x} \in \Sigma_{T_n} \right\} \quad (10)$$

Notice that the high-level problem (9) has a reduced size search space (from 2^E to $2^V \approx O(2^{\sqrt{E}})$), and the low-level subproblem corresponds to a single STP of a smaller subgraph $G[\mathbf{y}_n]$. Therefore, the original NP-hard problem (6) is decomposed into two smaller NP-hard problems. The overview of our hierarchical decomposition method for STPP is illustrated in Figure 2.

4.2 MDP Formulation and Hierarchical Policy for Steiner Tree Packing Problem

The high-level MDP \mathcal{M}_{hi} for STPP is based on a sequential decision making $\mathbf{y}_1, \dots, \mathbf{y}_N$ in equation 9. Formally, a state in the MDP is a tuple $s_t = (G, \mathcal{T}, S_t, t)$, where G is a weighted graph of the problem, \mathcal{T} the collection of set of terminals, and $S_t \subset V(G)$ is a partial solution constructed until the current timestep t via previous actions. An action a_t is to select a set of vertices $\mathbf{y}_t \subset V(G) \setminus S_t$ which includes a tree that spans the terminals $T_t \in \mathcal{T}$ as a subgraph of $G[\mathbf{y}_t]$. In turn, the subgraph $G[\mathbf{y}_t]$ is forwarded to the

low-level agent which solves STP on the given sub-graph by choosing the edges from $E(G[\mathbf{y}_t])$. Then, the high-level agent receives negative of the sum of the edge weights of the low-level solution $L(\mathbf{y}_t)$ as a reward, and appends the solution $L(\mathbf{y}_t)$ to the previous partial solution S_t .³ If the low-level solution $L(a_t)$ does not exist, or when any future choice of actions a_{t+1}, \dots, a_N leads to an infeasible solution of the given STPP, the high-level agent receives a large negative penalty $C < 0$.

In the prior decomposition method (i.e., TAM), $V(L(\mathbf{y}_t))$ and $V(G[\mathbf{y}_t])$ are the same. In other words, once nodes are selected as a high-level action, they are all used in the low-level solution, which prevents the next high-level policy from selecting any of the $V(L(\mathbf{y}_t))$ from the previous step. However, in the STPP, as shown in Figure 1, using all nodes that constitute the low-level problem, $V(G[\mathbf{y}_t])$, often leads to a low probability of finding a feasible solution. HCO reduces the likelihood of this occurring by removing only $V(L(\mathbf{y}_t))$ from $V(G)$ after getting the low-level solution. We defer detailed settings of our MDP formulation to the Appendix.

Model Architecture. Since STPP is a CO problem defined over a weighted graph, we use a graph neural network (GNN) to encode the state representation with the policy network π_θ and the value function V^{π_θ} that serves as a baseline for actor-critic methods (Konda and Tsitsiklis, 1999). Let G be the weighted graph with edge weights w_e as

³Here, the low-level sub-problem can again be defined by a MDP \mathcal{M}_{lo} , which is equivalent to the original decision making process (3) but on a smaller problem instance $\text{span}(\phi^{-1}(a_t))$.

an edge feature for each $e \in E(G)$. First, a D -dimensional node feature μ_v is computed for each node $v \in V$. Please refer to Appendix for our detailed choice of node and edge features. Then, the extracted features are encoded with graph attention network (GAT) (Veličković et al., 2018) and attention network (AT) (Vaswani et al., 2017). GAT aggregates the information across the *neighbors* in the graph to capture the local connectivity, but it is limited in modeling the long-range dependency (Vaswani et al., 2017). We overcome the limitation by using the attention network. The attention network captures the long-range dependency by encoding relation between *all* (i.e., ignores the graph structure) pairs of nodes. Global structures are further encoded via a graph embedding layer, which embeds particular subsets of node features into groups based on their characteristics.

We use behavioral cloning and reinforcement learning to train high-level policy and use a mathematical solver (i.e., MILP solver) for solving the low-level problem (10) in our implementation⁴.

5 EXPERIMENTS

5.1 Setting

Dataset. To evaluate the learning efficiency and generalization capacity of each algorithm, we created *feasibility-hard* STPP instances at various scales. Our instances included graphs of 40, 60, 80, and 100 nodes. For each graph category, we generated 50,000 instances for training, 1,000 for testing, and 100 for validation. Graphs were created using the Watts-Strogatz (WS) model (Watts and Strogatz, 1998) with mean node degree $k \sim \text{Uniform}(3, 4, 5, 6)$ and rewiring probability $\beta \sim \text{Uniform}(0, 1)$. Edge weights were uniformly random within $[0, 1]$. A subset of vertices was selected as terminals (see Section 3.2) to construct the STPP instances. To avoid unsolvable or trivially solvable instances, we designed a terminal selection algorithm ensuring instances were both solvable and non-trivial. We partitioned the random graph into $\mathcal{T} = N_{\text{type}}$ subgraphs using Lukes algorithm (Lukes, 1974), then chose N_{terminal} terminals per subgraph. Each subgraph ensured a spanning tree, but we filtered out trivial or unsolvable instances.

⁴Since our decomposition keeps the size of low-level subproblem small, we can run the MILP solver in a short time.

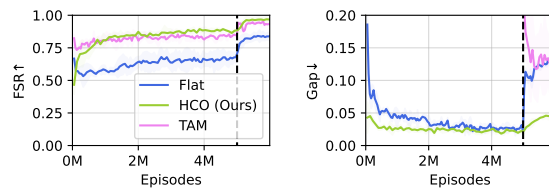


Figure 3: Training performance of the HCO for problem size $n = 40$. We pre-trained the agents via behavioral cloning until 5 millions episodes (i.e., vertical dotted line in the figure), and then finetuned via reinforcement learning afterwards. We report the performance averaged over 4 random seeds.

Baselines. We compare our model with following baselines. MILP- t utilizes the mixed integer linear programming (MILP) solver to identify the best solution within a specified time limit, denoted by t . Our implementation employs OR-Tools (Perron and Furnon, 2022) to achieve this. For a fair comparison, we set t to 1 second for MILP- t roughly matching the execution time of the compared methods. MILP- ∞ on the other hand, is the MILP solver operating without any time constraints, enabling it to find the optimal solution. PathFinder (McMurchie and Ebeling, 1995) is a heuristic algorithm designed to solve the STPP, as detailed in Section 2. We used a publicly available implementation of Lee et al. (2022), which includes two variations of low-level solvers: the shortest-path finding variant (PathFinder-SP) and the two-approximation variant (PathFinder-TA). Flat represents a non-hierarchical RL agent that constructs a solution by sequentially selecting nodes one by one, following the methodology outlined in studies Khalil et al.; Kool et al.. The Two-stage Divide Method (TAM) (Hou et al., 2022) addresses the STPP by decomposing it entirely in a single step, after which each sub-problem is addressed individually. This method differs from the HCO approach, which iteratively selects and solves one sub-problem at a time.

Training. We discovered that starting training with a random policy made it difficult to obtain feasible solutions, resulting in minimal learning signals. Therefore, for HCO training, we perform pre-training using behavioral cloning, followed by fine-tuning with IMPALA (Espeholt et al., 2018). The behavioral cloning data was generated using a solver OR-Tools, and training was conducted using cross-entropy loss. The training was conducted for 500K episodes to ensure convergence to some extent in all methods. We used the reinforcement learning framework RLlib for training with IMPALA. Hyperparameters were determined based on performance on the validation set, and the detailed process is described in Appendix.

Table 1: Result table for Steiner tree packing problem. Gap and FSR denote the average optimality gap and feasible solution ratio, respectively, while ET represents the average time taken to process a test instance. We report the performance averaged over 4 random seeds.

		$n = 40$			$n = 60$		
Method		Gap↓	FSR↑	ET (ms)↓	Gap↓	FSR↑	ET (ms)↓
Learning-based	HCO (Ours)	0.039	0.969	38	0.031	0.953	99
	TAM	0.105	0.885	17	0.128	0.914	46
	Flat	0.045	0.957	106	0.087	0.935	252
Heuristic	MILP-1s	0.000	1.000	127	0.000	0.982	501
	PathFinder-SP	0.112	0.974	5	0.165	0.990	8
	PathFinder-TA	0.116	0.966	19	0.147	0.974	48
Exhaustive	MILP-∞	0.000	1.000	125	0.000	1.000	532
		$n = 80$			$n = 100$		
Method		Gap↓	FSR↑	ET (ms)↓	Gap↓	FSR↑	ET (ms)↓
Learning-based	HCO (Ours)	0.054	0.932	124	0.056	0.892	246
	TAM	0.246	0.620	47	0.291	0.520	47
	Flat	0.087	0.902	529	0.062	0.905	679
Heuristic	MILP-1s	0.001	0.832	975	0.000	0.035	1007
	PathFinder-SP	0.150	0.976	20	0.155	0.970	35
	PathFinder-TA	0.149	0.965	115	0.150	0.954	170
Exhaustive	MILP-∞	0.000	1.000	1648	0.000	1.000	4685

Table 2: Generalization performance in terms of Common Instance Gap (CIG). The CIG measures the Gap averaged over the instances where all methods (HCO, TAM, Flat) found feasible solutions. Models are trained with $n = 40$ and are tested with $n = \{40, 60, 80, 100\}$. We report performance averaged over 4 random seeds.

	$n = 40$	$n = 60$
HCO (Ours)	0.033 (±0.001)	0.033 (±0.002)
TAM	0.090 (±0.007)	0.332 (±0.026)
Flat	0.065 (±0.012)	0.086 (±0.014)
	$n = 80$	$n = 100$
HCO (Ours)	0.041 (±0.003)	0.059 (±0.006)
TAM	1.017 (±0.113)	2.910 (±0.126)
Flat	0.120 (±0.014)	0.260 (±0.056)

Table 3: The average likelihood of decomposition policy selecting *trap nodes*, *optimal nodes*, and *redundant nodes* for test instances with $n = 40$ and $|\mathcal{T}| = 2$.

	Trap↓	Optimal↑	Redundant↓
HCO (Ours)	0.422	0.887	0.608
TAM	0.494	0.898	0.747

Evaluation. We use three metrics to evaluate the algorithm’s capability to minimize the cost and sat-

isfy the constraint. *Feasible solution ratio (FSR)* is the ratio of instances where a feasible (*i.e.*, constraint is satisfied) solution was found by the method. Since the solution cost can be computed only for a feasible solution, we also introduce the metric *optimality gap (Gap)*, measuring the average of the cost suboptimality in feasible solutions found: $\text{Gap} = \left(\frac{\text{algorithm cost}}{\text{optimal cost}} - 1 \right)$. Finally, *elapsed time (ET)* measures the average wall clock time taken to solve each instance in the test set. We report the performance averaged over four random seeds.

5.2 Result

Generalization to Unseen Instances with the Same Graph Size. Table 1 summarizes the performance of each method when the training and test sets consist of graphs of the *same* size but are mutually exclusive. The top three rows compare the learning-based algorithms: HCO, TAM, Flat. Overall, HCO is the most performant algorithm in terms of Gap and FSR. We observe that HCO consistently outperforms Flat with much less computation required (*i.e.*, smaller ET). Our hierarchical framework improves the sample efficiency of reinforcement learning by reducing the search space in high-level and low-level problems, leading to HCO’s superior performance in

Gap and FSR. Also, Flat sequentially solves problem by choosing the node one by one without decomposition, which requires more number of feed-forwards (*i.e.*, higher ET) per problem compared to HCO. As shown in Figure 1, TAM has a high probability of failing to obtain a feasible solution, resulting in a relatively low FSR. Additionally, we observed that learning signals from infeasible solutions significantly destabilize the training process. Rest of the table summarizes the performance of heuristic and exhaustive search-based methods. MILP-1s and MILP- ∞ performs search over solution space, where MILP-1s constrains the search based on computation time and MILP- ∞ performs search exhaustively. We note that MILP-based approaches excels in minimizing the cost once feasible solution is found, but suffers from finding any feasible solution especially when the problem size grows. This results in drastic degradation in FSR for MILP-1s and ET for MILP- ∞ for larger-sized problems ($n = 80$ and 100). When we compare PathFinder-based approaches to HCO, HCO is significantly advantageous in terms of Gap. We attribute the high FSR of PathFinder to their iterative algorithm, *negotiated-congestion avoidance*, which is tailored for finding feasible solutions in STPP.

Generalization to Unseen and Larger Instances.

To facilitate a clearer comparison, we introduce Common Instance Gap (CIG) metric, which evaluates the Gap exclusively on instances where every compared method identified a feasible solution. This approach ensures a fair comparison ground. We present the CIG performance, averaged across four distinct random seeds in the Table 2 (values in parentheses represent standard error). The results demonstrate HCO's superior performance over baseline methods across all tested instance sizes ($n = 40, 60, 80, 100$), with a notably wider performance margin against Flat in larger or more novel instances ($n = 80$ and 100). This underscores the bi-level formulation's effectiveness in enhancing generalization capabilities, even in more challenging scenarios.

Analysis of Decomposition-Based Models.

We further evaluate the decomposition performance of the decomposition-based models, HCO and TAM. When the model decomposes the given problem into sub-problems for each net, it must consider three types of nodes. First, the *trap node* is a critical part of another net's solution and *must not* be used for solving the current net. Second, the *optimal node* is included in the optimal solution. Third, the *redundant node* refers to nodes that do not fall into either of the previous categories. Generally, including redundant

nodes in the solution may increase the cost (*i.e.*, sub-optimal). Effective decomposition should avoid trap and redundant nodes while incorporating as many optimal nodes as possible. To assess this, we measure the average likelihood of each decomposition algorithm (*i.e.*, high-level policy) selecting trap, optimal, and redundant nodes. Our analysis was conducted on all instances with two nets (*i.e.*, $|\mathcal{T}| = 2$) from the $n = 40$ test instances. The results are presented in Table 3. Overall, TAM shows a higher likelihood of selecting any type of node compared to HCO. Notably, the likelihood of selecting optimal nodes is similar for both HCO and TAM. This indicates that while both methods are capable of effectively identifying nodes that contribute to the optimal solution, TAM is less successful at avoiding negative nodes (*i.e.*, trap and redundant nodes), which may lead to infeasible solutions or suboptimality. This is further supported by the results in Tables 1 and 2, where TAM shows a larger Gap and lower FSR due to its higher tendency to select redundant and trap nodes. This difference in performance can be attributed to the methods' approaches. Unlike TAM, which decomposes all problems at once, HCO re-evaluates and adjusts its decomposition after reviewing the results of the low-level problem, leading to more cautious decision-making.

Learning Curves. The learning curve for $n = 40$ is shown in Figure 3. The supervised pre-training improves both FSR and Gap of all the methods but the performance improvement plateaus around 4M steps. During RL finetuning, the FSR rapidly improves for all the methods while Gap worsens. The observed rise in FSR, indicating more feasible solutions, correlates with a growing gap. This suggests that instances solved later in training often need more training to reach optimality. Consequently, as FSR rises sharply, the model encounters many new instances for which it initially only finds suboptimal (*i.e.*, higher Gap) solutions, leading to an increase in Gap.

6 CONCLUSIONS

In this work, we proposed a novel hierarchical approach to address challenges of finding feasible solution for STPP. It uses latent mapping to decompose the solution search space, resulting in more sample-efficient learning due to separation of concerns and a smaller search space, and improved generalization from a homogeneous problem size for the low-level policy. We showed that the proposed decomposition framework is generally applicable to broader scope of combinatorial optimization problems. The effective-

ness of our method was evaluated on various sizes of STPP instances, demonstrating improved sample efficiency and generalization capability, outperforming heuristic, mathematical optimization, and learning-based algorithms designed for STPP.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2024-00410082).

REFERENCES

- Cappart, Q., Moisan, T., Rousseau, L.-M., Prémont-Schwarz, I., and Cire, A. A. (2021). Combining reinforcement learning and constraint programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3677–3687.
- Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416. PMLR.
- Fu, Z.-H., Qiu, K.-B., and Zha, H. (2021). Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7474–7482.
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32.
- Hou, Q., Yang, J., Su, Y., Wang, X., and Deng, Y. (2022). Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*.
- Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.
- Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.
- Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations*.
- Lee, K., Park, Y., Jeong, H.-S., Yoon, D., Hong, S., Sohn, S., Kim, M., Ko, H., Lee, M., Lee, H., Kim, K., Kim, E., Cho, S., Min, J., and Lim, W. (2022). ReSPack: A large-scale rectilinear steiner tree packing data generator and benchmark. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*.
- Li, S., Yan, Z., and Wu, C. (2021). Learning to delegate for large-scale vehicle routing.
- Lukes, J. A. (1974). Efficient algorithm for the partitioning of trees. *IBM Journal of Research and Development*, 18(3):217–224.
- Ma, Y., Hao, X., Hao, J., Lu, J., Liu, X., Xialiang, T., Yuan, M., Li, Z., Tang, J., and Meng, Z. (2021). A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. *Advances in Neural Information Processing Systems*, 34:23609–23620.
- McMurchie, L. and Ebeling, C. (1995). Pathfinder: A negotiation-based performance-driven router for fpgas. In *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*, pages 111–117.
- Nowak-Vila, A., Folqué, D., and Bruna, J. (2018). Divide and conquer networks.
- Perron, L. and Furnon, V. (2022). Or-tools.
- Song, J., Lanka, R., Yue, Y., and Dilkina, B. (2020). A general large neighborhood search framework for solving integer linear programs.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations*.
- Wang, R., Hua, Z., Liu, G., Zhang, J., Yan, J., Qi, F., Yang, S., Zhou, J., and Yang, X. (2021). A bi-level framework for learning to solve combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 34:21453–21466.
- Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of “small-world” networks.
- Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., and Li, F. (2024). Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20284–20292.

APPENDIX

Combinatorial Optimization as Markov Decision Process

In this section, we provide a full illustration of a sequential optimization process for a general CO problem. Our goal is to design a corresponding MDP for equation 3. Intuitively, the sequential decision making process in equation 3 can be thought of choosing for each timestep t an action $\mathbf{x}_t \in X_t$, until we have a full solution $\mathbf{x} = \sum_t \mathbf{x}_t$. However, note that we are constructing X_1, \dots, X_H *sequentially*, *i.e.*, we do not have the subspace decomposition $X = X_1 \oplus \dots \oplus X_H$ beforehand. Hence, assume we have constructed X_1, \dots, X_t until the current timestep t . Let us define W_t the remaining subspace that are yet to be decomposed, *i.e.*, $X = X_1 \oplus \dots \oplus X_t \oplus W_t$. Then, the sequential decision making is equivalent to choosing for each timestep t a subspace $X_t \leq W_t$ and consequently an action $\mathbf{x}_t \in X_t$, until we have a trivial subspace $W_t = \{\mathbf{0}\}$.

Sequential decision making process for the bi-level decomposition in equation 7 and 8 can be formulated similarly. The key is to construct the subspace decomposition $Y = Y_1 \oplus \dots \oplus Y_N$ sequentially, while obtaining the partial solution for the original problem from the low-level sub-problem (8) simultaneously.

MDP Formulation

Below we provide a full description of our MDP formulation for STPP in section 4.2.

State. The state s_t of the high-level MDP \mathcal{M}_{hi} consists of a graph G , collection of sets of terminals \mathcal{T} , a partial solution $S_t \subset V(G)$ constructed until timestep t , (*i.e.*, the nodes of the disjoint trees that span terminals T_1, \dots, T_{t-1}) and the current timestep t . The graph G provides general information of the problem instance, *i.e.*, the connectivity of the graph via edges. In practice, the graph G can be represented by the adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$, where a_{ij} takes the edge weight w_{ij} . The information can be further encoded via message passing layers of GAT and AT in GNN. \mathcal{T} , S_t and t provide node features for the current timestep, and are essential for generating a graph embedding. From the state information s_t , we extract the node features of the graph for further encoding via the GNN model. For a node $v \in V$, we denote the node features of the vertex v as $\mathbf{x}_v := (x_o, x_\tau, x_d) \in \mathbb{Z}^3$. The first node feature $x_o \in \{0, 1\}$ denotes whether a node v is included in the current partial solution or not. If v is selected as part of the partial solution, we define

$x_o = 1$, otherwise, it is 0. The second node feature $x_\tau \in \{0, 1, \dots, N_{type}\}$ indicates the terminal type (*i.e.*, $x_\tau = k$ if and only if $v \in T_k$), where the indices are labeled in the order that the high-level MDP solves for. Non-terminal nodes are assigned a value of 0. The last feature, x_d , denotes the degree of a node $v \in V$. The edges of the graph are also assigned with edge features. We only use the edge weights as the edge features in this paper. Our choice of node and edge features is summarized in Table 4, 5.

Action. The action a_t of a high-level MDP \mathcal{M}_{hi} at timestep t is to select a set of vertices $\mathbf{y}_t \subset V(G) \setminus S_t$, where S_t is the partial solution constructed until the previous timesteps. Intuitively, the action a_t is to select a node *set* that includes all terminals of the current type at timestep t . The selected set of nodes should create a subgraph $G[a_t]$ of G generated by the nodes a_t , and should correspond to a STP instance (with single type of terminals). The low-level agent then computes a minimum weight tree that spans all terminals of current type from the subgraph $G[a_t]$. In practice, we further assist the agent by designing the MDP environment for \mathcal{M}_{hi} in a way such that the terminal nodes of the current type are automatically selected by the environment internally. Hence, the action a_t will result in a subgraph $G[a_t \cup V(T_t)]$ instead of $G[a_t]$.

Reward. Designing a reward with an optimality guarantee is a non-trivial task. In particular, we wish to construct a reward where all feasible solutions result in higher reward than those of any infeasible solutions. Also, the feasible solutions with *better* solution quality (*i.e.* the objective being closer to optimal solution) should be assigned a higher reward. Hence, given a final solution \mathbf{x} of a CO problem (1), a reward with an optimality guarantee can be compactly formulated as $r(\mathbf{x}) = c \cdot \mathbf{1}_{\mathcal{F}}(\mathbf{x}) - f(\mathbf{x})$, where $c \geq \sup_{\mathbf{x} \in X} f(\mathbf{x})$, f is the objective function of problem (1), and $\mathbf{1}_{\mathcal{F}}(\cdot)$ is an indicator function. Note that this form of the reward is equivalent to what is described in Section 3.1. Achieving the maximal return will result in the same optimal policy⁵. Finally, recall that our objective f is linear in X . For each partial solution \mathbf{x}_t constructed at timestep t , we are able to decompose our reward function as follows.

$$r(\mathbf{x}_t) = c \cdot \mathbf{1}_{\mathcal{F}} \left(\sum_{k=1}^t \mathbf{x}_k \right) - f(\mathbf{x}_t) \quad (11)$$

Transition. Our transition in MDP is deterministic; the change in the partial solution alters a node feature

⁵Providing *incentives* for $\mathbf{x} \in \mathcal{F}$ instead of a penalty when $\mathbf{x} \notin \mathcal{F}$ scales all rewards to be non-negative.

Table 4: Node features.

Notation	Value	Description
x_o	$\in \{0, 1\}$	Is partial solution
x_τ	$\in \{0, 1, 2, \dots, N_{\text{type}}\}$	Terminal type
x_d	$\in \mathbb{Z}$	Node degree

Table 5: Edge feature.

Notation	Value	Description
w_e	$\in \mathbb{R}$	Edge weight(=cost)

x_o from 0 into 1, which results in different node and graph embeddings from GNN.

Termination. Our STPP environment is terminated when it is not able to generate a feasible STPP solution or when a feasible solution is found. The cases where generating a feasible STPP solution includes (1) no possible actions remaining, (2) any choice of actions in future timestep inevitably results in an infeasible solution, or (3) the choice of action a_t in a high-level MDP \mathcal{M}_{hi} results in an unsolvable STP instance $G[a_t]$.

GNN Architecture

Encoder. Given a graph G , we first extract a D -dimensional node embedding μ_v for each node $v \in V$, where D denotes the number of features, as provided in Section 6. Note that we use $D = 3$, with $\mathbf{x}_v = (x_o, x_\tau, x_d)$ representing the features as described in Table 4. Let $\rho: \mathbb{R}^D \rightarrow \mathbb{R}^{D-p}$ be a fixed vectorization mapping of a given node feature \mathbf{x}_v , and let $\theta_0: \mathbb{R}^{D-p} \rightarrow \mathbb{R}^p$ be a linear mapping. We obtain the initial node embedding μ_v as follows.

$$\mu_v = \text{ReLU}(\theta_0(\rho(\mathbf{x}_v))). \quad (12)$$

Next, we encode the node embeddings $\mathbf{M} := (\mu_1, \dots, \mu_{|V|}) \in \mathbb{R}^{|V| \times p}$ via graph attention network (GAT) and attention network (AT). Formally, let $\Theta_i: \mathbb{R}^{n_h \times p} \rightarrow \mathbb{R}^{2p}$ and $\theta_i: \mathbb{R}^{2p} \rightarrow \mathbb{R}^p$ for $i = 1, \dots, l$ be linear mappings, where n_h denotes the number of heads of GAT. For convenience, we slightly overload the notation, writing $\Theta_i(\mathbf{M}) := (\Theta_i(\mu_1), \dots, \Theta_i(\mu_{|V|}))$, and use a similar notation for θ_i . We recursively encode the node embedding as follows

$$\mathbf{M}^{(i-1)'} = \text{AT}(\Theta_i(\text{GAT}(\mathbf{M}^{(i-1)}; G)); G) \quad (13)$$

$$\mathbf{M}^{(i)} = \theta_i(\text{ReLU}(\mathbf{M}^{(i-1)} \parallel \mathbf{M}^{(i-1)})) \quad (14)$$

for each $i = 1, \dots, l$. Here, we define $\mathbf{M}^{(0)} \equiv \mathbf{M}$, and write $(\parallel \cdot)$ for $\text{CONCATENATE}(\cdot, \cdot)$. We denote

our graph encoder function **Enc** succinctly as follows, omitting some details for brevity.

$$\mathbf{Enc}(\cdot; G) := \overbrace{(\text{AT} \circ \text{GAT}) \circ \dots \circ (\text{AT} \circ \text{GAT})}^{l \text{ times}}(\cdot; G) \quad (15)$$

Graph Embedding, Logit, and Probability.

Given the node embedding of the last layer $\mathbf{M}^{(l)}$, we obtain the embedding for the entire graph μ^G . Instead of simply averaging over all the nodes, we enrich the graph embedding by grouping the nodes into three subsets based on their characteristics: current terminal T_t , partial solution S_t , and non-terminal nodes $\bar{V} := V \setminus \bigcup_{T \in \mathcal{T}} T$. Then, the embeddings are averaged within each subset, concatenated, and projected to obtain the graph embedding μ^G . Formally, the graph embedding layer **Emb** is written as follows.

$$\mathbf{Emb}(\cdot; t) := \Psi(\cdot, T_t) \parallel \Psi(\cdot, S_t) \parallel \Psi(\cdot, \bar{V}) \quad (16)$$

where $\Psi(\mathbf{M}^{(l)}, A)$ performs the average pooling over the set of node embeddings that belong to A . The graph embedding μ^G is obtained as $\mu^G = \mathbf{Emb}(\mathbf{M}^{(l)}; t)$. Finally, the logit value, which is used to compute the probability of choosing the node for each node $v \in V$ is computed as follows.

$$\text{logit}_v = w_4(w_3(\text{ReLU}(w_1(\mu^G) \parallel w_2(\mu_v^{(l)}))) + \mu_v^{(0)}) \quad \forall v \in V \quad (17)$$

$$p_v = \text{softmax}(\text{logit}_v) \quad \forall v \in V \quad (18)$$

where $w_1: \mathbb{R}^{3p} \rightarrow \mathbb{R}^p$, $w_2: \mathbb{R}^p \rightarrow \mathbb{R}^p$, $w_3: \mathbb{R}^{2p} \rightarrow \mathbb{R}^p$, and $w_4: \mathbb{R}^p \rightarrow \mathbb{R}^2$ are linear functions.

Value Function. The value function V^{π_θ} uses a model that has a similar GNN architecture with a simple multi layer perceptron MLP that sequentially projects $(\mathbb{R}^{3p} \rightarrow \mathbb{R}^{3p} \rightarrow \mathbb{R}^p \rightarrow \mathbb{R}^1)$, and does not share weights with the policy network.

$$V^{\pi_\theta}(s_t) = \text{MLP}(\mathbf{Emb}(\mathbf{Enc}(\mathbf{M}; G); t)). \quad (19)$$

Hyperparameters

We set the hyperparameters based on the performance of Flat and considered the following hyperparameter ranges for the 40-node experiment:

- $p = \{32, 64, 128, 256\}$
- $l = \{3, 4, 5\}$
- number of heads = $\{4, 8\}$
- IL learning rate = $\{1e-3, 1e-4, 1e-5, 1e-6\}$
- weight decay = $\{1e-6, 5e-7\}$

- RL learning rate = $\{1e-3, 1e-4, 1e-5, 1e-6\}$
- entropy coefficient = $\{1, 1e-1, 1e-2\}$
- value loss coefficient = $\{1e-1, 1, 3, 5\}$

Thirty hyperparameter combinations were randomly selected from the given ranges, and the combination that performed best on the validation set was used. This parameter combination was consistently used for all methods. The GNN model uses a hidden dimension of $p = 128$ and $l = 5$ for both GAT and AT encoder layers. Both GAT and AT use 8 heads, with a dropout rate of 0.5 in IL, but dropout is not used in RL training. A batch size of 64 is used, and the learning rate is initialized to 10^{-4} , decreasing by 0.99 per epoch. A fine-tuned value of 5×10^{-7} is used for weight decay. To prevent divergence in learning, the gradient norm is clipped to 1. HCO trains for 100 epochs, using 1 epoch to update the model with BC data at every step and in every episode. In the RL phase, a batch size of 30, a learning rate of 10^{-6} , a discount factor of 0.99, an entropy coefficient of 0.01, and a value function loss coefficient of 5 are used. The number of workers used in IMPALA is set to 30.

Training and Evaluation

For imitation learning, we first collect the demonstration data using the optimal solver, MILP, as expert policy. The expert policy π^{expert} is defined as $\pi^{\text{expert}}(a_t|\cdot) = 1$ if $a_t \in \mathcal{A}_t^*$ and $\pi^{\text{expert}}(a_t|\cdot) = 0$ otherwise. The nodes that are not selected by the expert are excluded from the loss calculation. HCO uses cross-entropy loss for BC training, and one epoch is defined as updating HCO for every step of every instance. During the evaluation phase, the softmax function is applied to the logit values of each node to obtain probabilities, and nodes with probability values exceeding 0.5 are selected as high-level actions. The training and evaluation are carried out on a single GPU, comprising an AMD EPYC 7R32 CPU and NVIDIA A10G GPU.