A PROOF FOR THEOREM 1

Consider $\pi_\theta(\vec{a}|s_1)$ as the distribution over the action trajectory from a state s_1 which describes the problem context. Let $\vec{\mathcal{A}}_{\boldsymbol{x}}$ denote the space of action-trajectories associated with the solution $\boldsymbol{x}$.

$$\begin{aligned}
\mathcal{H}(\pi_\theta(\vec{a}|s_1)) &= -\sum_{\vec{a}\in\vec{\mathcal{A}}} \pi_\theta(\vec{a}|s_1)\log\pi_\theta(\vec{a}|s_1) \\
&= -\sum_{\boldsymbol{x}\in\mathcal{X}}\sum_{\vec{a}\in\vec{\mathcal{A}}_{\boldsymbol{x}}} \pi_\theta(\vec{a}|s_1)\log\pi_\theta(\vec{a}|s_1) \\
&= -\sum_{\boldsymbol{x}\in\mathcal{X}}\sum_{\vec{a}\in\vec{\mathcal{A}}_{\boldsymbol{x}}} p_{\text{sym}}(\vec{a}|\boldsymbol{x})p(\boldsymbol{x}|s_1)\left(\log p_{\text{sym}}(\vec{a}|\boldsymbol{x}) + \log p(\boldsymbol{x}|s_1)\right) \\
&= \mathcal{H}(p(\boldsymbol{x}|s_1)) + \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}|s_1)}\mathcal{H}(p_{\text{sym}}(\vec{a}|\boldsymbol{x})) \\
&\leq \mathcal{H}(p(\boldsymbol{x}|s_1)) + \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}|s_1)}\mathcal{H}(U_{\boldsymbol{x}}(\vec{a}|\boldsymbol{x})),
\end{aligned}$$

where $U_{\boldsymbol{x}}(\vec{a}|\boldsymbol{x})$ is a uniform distribution over action-trajectories associated with the solution $\boldsymbol{x}$. The third equality stems from the fact that $\pi_\theta(\vec{a}|s_1) = \pi_\theta(\vec{a},\boldsymbol{x}|s_1)$ since $\boldsymbol{x}$ is fixed given $\vec{a}$. One can show that the final upper-bound is the entropy of distribution obtained from replacing $p_{\text{sym}}(\vec{a}|\boldsymbol{x})$ by $U_{\boldsymbol{x}}(\vec{a}|\boldsymbol{x})$.

B IMPLEMENTATION DETAILS

B.1 PROXIMAL POLICY OPTIMIZATION (PPO)

We use AM architecture (Kool et al., 2018) on TSP and Devformer architecture (Kim et al., 2023) on DPP for parameterizing compositional policy $\pi(\boldsymbol{x}|s_1) = \prod_{t=1}^{N} \pi(a_t|s_t)$. Then, we implement based on the following equation as follows:

$$\mathcal{L}(\boldsymbol{x}; s_1) = \min \left[A(\boldsymbol{x}; s_1) \frac{\pi(\boldsymbol{x}|s_1)}{\pi_{\text{old}}(\boldsymbol{x}|s_1)}, A(\boldsymbol{x}; s_1) \text{clip} \left(\frac{\pi(\boldsymbol{x}|s_1)}{\pi_{\text{old}}(\boldsymbol{x}|s_1)}, 1-\epsilon, 1+\epsilon \right) \right],$$
$$A(\boldsymbol{x}; s_1) = R(\boldsymbol{x}; s_1) - V(s_1),$$

where R stands for reward function and V stands for value function. Since we implement PPO on compositional MDP setting, we train value function in the context of s_1 by following actor-critic implementation of Kool et al. (2018).

Hyperparameters. We systematically investigate a range of hyperparameter combinations involving different baselines ([rollout, critic]), various values for clipping epsilon ([0.1, 0.2, 0.3]), and numbers of inner loops ([5, 10, 20]). Our observations reveal that the critic baseline consistently enhances training stability across all tasks, leading to reduced variation when modifying the training seeds. The best configurations for each task are provided in Table 4.

Table 4: Hyperparameter configurations for PPO.

	TSP	Chip-package PDN	HBM PDN
Baseline	critic	critic	critic
Eps. clip	0.2	0.1	0.2
Number of inner loops k	5	20	10

B.2 GENERATIVE FLOW NETWORK (GFLOWNET)

Similar to the PPO implementation, we employ the Attention Model (AM) architecture (Kool et al., 2018) on the Traveling Salesman Problem (TSP), and the DevFormer architecture (Kim et al., 2023) on DPP, for parameterizing the compositional forward policy $P_F(\tau|s_1) = \prod_{t=1}^{N} P_F(a_t|s_t)$. Subsequently, we configure the backward policy P_B as a uniform distribution for all possible parent nodes, following the methodology outlined in (Malkin et al., 2022). Lastly, we parameterize $Z(s_1)$ using a two-layer perceptron with ReLU activation functions, where the number of hidden units matches the embedding dimension of the AM or DevFormer. This two-layer perceptron takes input from the mean of the encoded embedding vector obtained from the encoder of the AM or DevFormer and produces a scalar value to estimate the partition function.

To train the GFlowNet model, we use trajectory balance loss introduced in Malkin et al. (2022) as follows:

$$\mathcal{L}(\tau; s_1) = \left(\log \left(\frac{Z(s_1) P_F(\tau|s_1)}{e^{-\beta E(\boldsymbol{x}; s_1)} P_B(\tau|s_1)} \right) \right)^2 \tag{3}$$

The trajectory τ includes a terminal state represented as $\boldsymbol{x}$. Subsequently, we employ an on-policy optimization method to minimize Eq. (3), with trajectories τ sampled from the training policy P_F. In this context, $E(\boldsymbol{x}; s_1)$ represents the energy, which is essentially the negative counterpart of the reward $R(\boldsymbol{x}; s_1)$. The hyperparameter β plays the role of temperature adjustment in this process.

Hyperparameters. We explore a spectrum of hyperparameter combinations, varying β ([5, 10, 20]) and numbers of inner loops ([2, 5, 10]). The best configurations for each task are provided in Table 5.

Table 5: Hyperparameter configurations for GFlowNet.

	TSP	Chip-package PDN	HBM PDN
β	20	10	10
Number of inner loops k	10	2	2

C EXPERIMENTAL DETAILS

C.1 TRAVELING SALESMAN PROBLEMS (TSP)

Since we employ the AM architecture, we use the same hyperparameters for the model architecture and training parameters except for the batch and epoch data sizes.[3] Initially, the Attention Model (AM) employed a batch size of 512 and an epoch data size of 1,280,000. Notably, the evaluation of the greedy rollout baseline was conducted every epoch. When the number of available training samples is constrained, utilizing a smaller batch size and epoch data size becomes advantageous. Consequently, we adjusted these parameters to be 100 for batch size and 10,000 for epoch data size. In symmetric self-distillation (Step B), the distillation coefficients are meticulously set to scale the SSD loss. As a rough guideline, we establish a coefficient that renders the SSD loss approximately 10 to 100 times smaller than the RL loss. Additionally, for the number of symmetric transformations (L in Eq. (1)) is set as the number of inner loops. See Table 6 in details.

Table 6: Distillation coefficient and the number of symmetric transformations in TSP.

	A2C	PG-Rollout	PPO	GFlowNet
Distillation coefficient	0.001	0.001	0.00001	0.1
L	1	1	5 ($= k$)	10 ($= k$)

C.2 DECAP PLACEMENT PROBLEMS (DPP)

Similar to the experiments on TSP, we follow the setting of DevFormer.[4] We set the batch size as 100 and epoch data size as 600. Note that the maximum number of reward calls is set 15K,, a considerably smaller limit compared to TSP. Regarding the distillation coefficient and the number of symmetric transformations, we maintain consistency with the principles applied in the TSP experiments as follows:

Table 7: Distillation coefficient and the number of symmetric transformations in DPP tasks.

	A2C	PG-Rollout	PPO	GFlowNet
Distillation coefficient	0.01	0.01	0.01	0.1
L	1	1	20 ($= k$)	2 ($= k$)

C.3 PRACTICAL MOLECULAR OPTIMIZATION (PMO)

We basically follow the experimental setting (e.g., batch size) in the practical molecular optimization (PMO) benchmark.[5] In the symmetric self-distillation step, we utilize reward-prioritized sampling for the online buffer, which contains molecules generated during online learning. For the REINVENT, where the replay buffer is already incorporated, we set the number of distillation samples equal to the replay buffer size, i.e., 24, and the distillation coefficient to 0.001. Regarding the GFlowNet method, we configure the number of distillation samples to match the batch size of 64. Furthermore, we set the distillation coefficient to 1.0, given that the RL loss in GFlowNet is much higher compared to REINVENT.

[3] AM: `https://github.com/wouterkool/attention-learn-to-route`

[4] DevFormer: `https://github.com/kaist-silab/devformer`

[5] Practical molecular optimization: `https://github.com/wenhao-gao/mol_opt`

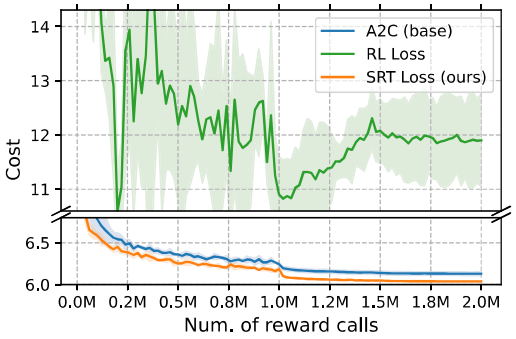


Figure 8: Validation cost over computation budget on TSP50.

D ADDITIONAL EXPERIMENTS

D.1 ABLATION STUDY FOR LOSS FUNCTIONS ON TSP50

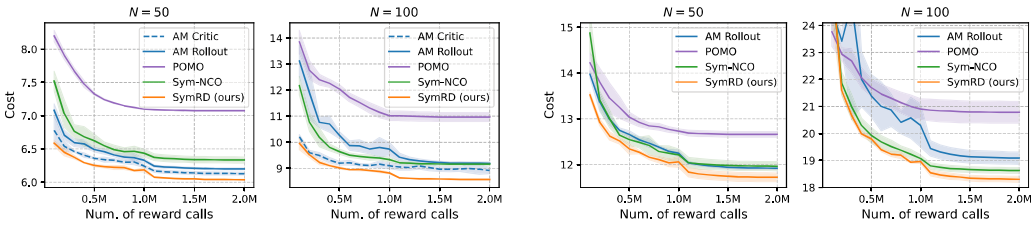
This subsection provide the results of ablation study for using imitation loss in symmetric replay training. The experiments are conducted with the same RL loss (denoted as 'RL Loss') and the proposed imitation loss (denoted as 'SRT Loss'). Since we employ the A2C as the base, RL loss is as follows:

$$\mathcal{L}_{RL} = \frac{1}{B}\sum_{i=1}^{B}(R(\boldsymbol{x}|s_1) - V(s_1))\log \pi_\theta(\vec{a}|s_1),$$

where B is batch size. As shown in Figure 8, it is evident that symmetric replay training with RL loss exhibits instability. This is a natural consequence since the symmetric trajectories often diverge significantly from the current policy.

Table 8: Experimental results on sample efficient Euclidean CO problems.

	Method	$N = 50$		$N = 100$	
		$K = 200$K	$K = 2$M	$K = 200$K	$K = 2$M
TSP	AM Critic	6.541 ± 0.075	6.129 ± 0.021	9.600 ± 0.090	8.917 ± 0.115
	AM Rollout	6.708 ± 0.077	6.199 ± 0.014	11.891 ± 1.008	9.193 ± 0.053
	POMO	7.910 ± 0.055	7.074 ± 0.010	12.766 ± 0.358	10.964 ± 0.171
	Sym-NCO	7.035 ± 0.209	6.334 ± 0.045	10.776 ± 0.362	9.159 ± 0.056
	SRT (ours)	$\mathbf{6.450 \pm 0.053}$	$\mathbf{6.038 \pm 0.005}$	$\mathbf{9.521 \pm 0.098}$	$\mathbf{8.573 \pm 0.019}$
CVRP	AM Rollout	13.366 ± 0.199	11.921 ± 0.026	23.414 ± 0.238	19.088 ± 0.232
	POMO	13.799 ± 0.310	12.661 ± 0.065	22.939 ± 0.245	20.785 ± 0.403
	Sym-NCO	13.406 ± 0.204	12.215 ± 0.124	21.860 ± 0.422	18.630 ± 0.106
	SRT (ours)	$\mathbf{12.922 \pm 0.071}$	$\mathbf{11.721 \pm 0.093}$	$\mathbf{21.582 \pm 0.149}$	$\mathbf{18.304 \pm 0.109}$



(a) TSP (b) CVRP

Figure 9: Validation cost over computation budget on euclidean CO problems.

D.2 Experiments on various synthetic CO problems

The experiments in this section cover various sample-efficient tasks in Euclidean and non-Euclidean combinatorial optimization. Note that we assume the expensive black-box reward function in sample-efficient tasks. In Euclidean CO tasks, the features of variables, such as their two-dimensional coordinates, satisfy Euclidean conditions (e.g., cost coefficients are defined as Euclidean distances). On the other hand, non-Euclidean CO problems lack these constraints, necessitating the encoding of higher-dimensional data, such as a distance matrix.

D.2.1 Euclidean CO problems

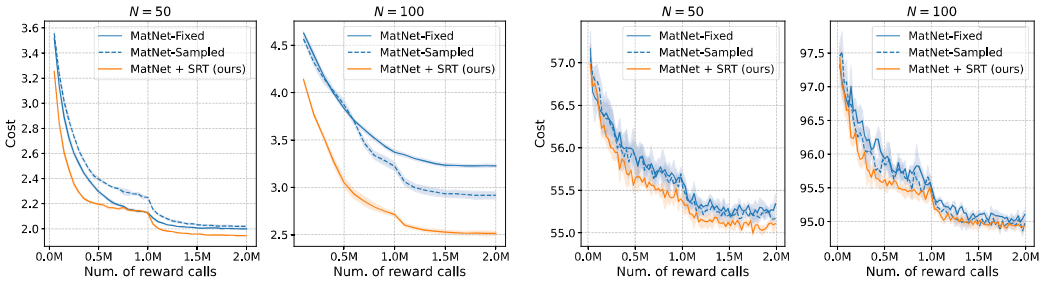
Experimental settings. We select two representative routing tasks – the travelling salesman problem (TSP) and the capacitated vechicle routing problem (CVRP) with 50 and 100 customers. The CVRP assumes multiple salesmen (i.e., vehicles) with limited carrying capacity; thus, if the capacity is exceeded, the vehicle must return to the depot. For base DRL methods, we employ the best-performing DRL methods, AM for TSP and Sym-NCO for CVRP. We follow reported hyperparameters for the model in their original paper.[6]

Results. The results in Table 8 and Fig. 9 indicate that SRT consistently outperforms baseline methods in terms of achieving the lowest cost over the training budget. Note that ours employs the AM with critic baseline for TSP and Sym-NCO with the reduced number of augmentations for CVRP. As depicted in Table 8, the most significant improvement over the base DRL models is observed in TSP100, with a percentage decrease of 3.86%, and CVRP50, with a percentage decrease of 4.04%. While POMO and Sym-NCO consider the symmetric nature of CO, the required number of samples cancels out the benefits. In contrast, our method utilizes the symmetric pseudo-labels generated via the training policy for free, enabling the policy to explore the symmetric space without increasing the number of required samples. As a result, SRT successfully improves sample efficiency.

[6]Sym-NCO: `https://github.com/alstn12088/Sym-NCO`

Table 9: Experimental results on sample efficient non-Euclidean CO problems.

		$N = 50$		$N = 100$	
	Method	$K =$ 200K	$K =$ 2M	$K =$ 200K	$K =$ 2M
ATSP	MatNet-Fixed	3.139 ± 0.024	2.000 ± 0.002	4.400 ± 0.040	3.227 ± 0.016
	MatNet-Sampled	3.235 ± 0.021	2.019 ± 0.005	4.324 ± 0.036	2.915 ± 0.040
	SRT (ours)	$\mathbf{2.845 \pm 0.039}$	$\mathbf{1.945 \pm 0.003}$	$\mathbf{3.771 \pm 0.012}$	$\mathbf{2.513 \pm 0.022}$
FSSP	MatNet-Fixed	56.350 ± 0.170	55.341 ± 0.118	96.461 ± 0.206	95.107 ± 0.072
	MatNet-Sampled	56.347 ± 0.234	55.172 ± 0.032	96.256 ± 0.140	94.978 ± 0.055
	SRT (ours)	$\mathbf{56.104 \pm 0.125}$	$\mathbf{55.110 \pm 0.061}$	$\mathbf{96.030 \pm 0.132}$	$\mathbf{94.934 \pm 0.051}$



(a) ATSP (b) FFSP

Figure 10: Validation cost over computation budget on non-Euclidean CO problems.

D.2.2 Non-Euclidean CO problems

Experimental settings. Based on the work of Kwon et al. (2021), we have selected two benchmark tasks, namely the asymmetric TSP (ATSP) and flexible flow-shop scheduling problems (FSSP). The ATSP is non-Euclidean TSP where the distance matrix could be non-symmetric, i.e., $\text{dist}(i, j) \neq \text{dist}(j, i)$, where i and j indicate cities. The FSSP is an important scheduling problem that assigns jobs to multiple machines to minimize total completion time. As a baseline, we employ Matrix Encoding Network (MatNet) proposed to solve non-Euclidean CO.[7] We compare ours with two versions of MatNet: MatNet-Fixed and MatNet-Sampled. MatNet-Fixed, the original version, explores N heterogeneous starting points of trajectories, while MatNet-Sampled explores less than N number of multiple trajectories with sampling strategy.

Results. The superior performance of SRT over MatNet-Fixed and MatNet-Sampled is demonstrated in both Table 9 and Fig. 10. We employ MatNet-Sampled as a base DRL method for both tasks and use the same number of multi-starting in ours and MatNet-Sampled. Notably, SRT outperforms MatNet-Sampled by a significant margin in the case of ATSP, with a performance gap of about 12% at $N = 100$, $K =$ 200K, where SRT achieves 3.771 and MatNet-Sampled achieves 4.324.

[7] MatNet: `https://github.com/yd-kwon/MatNet`