The Supplementary File of Learning Encodings for Constructive Neural Combinatorial Optimization Needs to Regret

Rui Sun[1]*, Zhi Zheng[1]*, Zhenkun Wang[2]†

A Model Structure

The proposed LCH-Regret is a model-agnostic modification and the model structure of LCH-Regret is always similar to its original LCH models. So, we will focus on the partitions we added for the regret mechanism.

A.1 Encoder

The encoder structure is the same as Kool et al. [1], Kwon et al. [2], meaning AM-Regret and MatNet-Regret have 3 attention layers while POMO-Regret has 6. From the d_x-dimensional input features x_i (for TSP d_x = 2), the encoder computes initial d_k-dim node embeddings $h_i^{(0)}$ by a learnable linear projection $h_i^{(0)} = W^x x_i + b^x$, $d_k = 128$. Then, the embeddings are updated by L attention layers [3]. Each attention layer has a multi-head attention layer (MHA), a feed-forward layer (FF), and normalization layers (Norm). The l-th attention layer can be calculated as follows:

$$\begin{aligned}\hat{\boldsymbol{h}}_i^{(l)} &= \textbf{Norm}^{(l)}(\boldsymbol{h}_i^{(l-1)} + \textbf{ATT}_i^{(l)}(\boldsymbol{h}_i^{(l-1)}, \boldsymbol{H}_j^{(l-1)})),\\ \boldsymbol{h}_i^{(l)} &= \textbf{Norm}^{(l)}(\hat{\boldsymbol{h}}_i^{(l)} + \textbf{FF}^{(l)}(\hat{\boldsymbol{h}}_i^{(l)})),\end{aligned} \tag{1}$$

where ATT$(\cdot)$ indicates the self-attention operation [3] and $l \in \{1, 2, \cdots, L\}$. As in the original setups, the normalization method used in AM-Regret is the Batch norm [4] while the normalization method used in POMO-Regret is the Instance norm [5]. The L is set to 3 in AM-Regret and 6 in POMO-Regret. The $\boldsymbol{h}_i^{(L)} \in \mathbb{R}^{d_k}$ represents the encoding of node i.

For MatNet [6], there is only a difference in encoder compared to AM and POMO. It proposes a novel encoder to handle matrix inputs.

In this paper, node encodings means $\boldsymbol{h}_i^{(L)}, i \in \{1, \cdots, n\}$.

A.2 Decoder

The decoder also utilizes the attention mechanism to calculate the probability of each node. It computes the probability $\pi_\theta(a_t|\boldsymbol{x}_t, c)$ in a Markov Decision Process with a given instance c and a partial solution $\boldsymbol{x}_t$. In LCH-Regret, encodings consist the node encodings $\boldsymbol{h}_i^{(L)}, i \in \{1, 2, \cdots, n\}$ and the Regret encoding $\boldsymbol{h}_R$.

The context embedding represents the information of the partial solution x_t. In AM [1], the context embedding $\boldsymbol{h}_c$ is

$$\boldsymbol{h}_c = \begin{cases} [\bar{\boldsymbol{h}}^{(L)}, \boldsymbol{h}_{\boldsymbol{x}_{t-1}}^{(L)}, \boldsymbol{h}_{\boldsymbol{x}_1}^{(L)}] & t > 1 \\ [\bar{\boldsymbol{h}}^{(L)}, \boldsymbol{v}_1, \boldsymbol{v}_f] & t = 1 \end{cases}, \tag{2}$$

*These authors contributed equally.

†Corresponding author.

where $\bar{\boldsymbol{h}}^{(L)} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{h}_i^{(L)}$ is a graph information and $\boldsymbol{v}_1, \boldsymbol{v}_f$ are learnable placeholders. The proposed AM-Regret adopts the setup of context embedding (the $\boldsymbol{h}_R$ will never be involved in it).

For POMO and MatNet, the context embedding is

$$\boldsymbol{h}_c = \boldsymbol{h}^{(L)}_{\boldsymbol{x}_{t-1}} + \boldsymbol{h}^{(L)}_{\boldsymbol{x}_1}. \tag{3}$$

POMO-Regret and MatNet-Regret also adopt the same setup of context embedding, which means that the Regret encoding $\boldsymbol{h}_R$ will never be involved in $\boldsymbol{h}_c$.

The following glimpse layer calculates the mutual attention between query, keys, and values, the attention result $\boldsymbol{q}_l$ for logit calculation is

$$\begin{gathered}\boldsymbol{q} = W^Q\boldsymbol{h}_c, \quad \boldsymbol{k} = W^K\left[\boldsymbol{h}_1^{(L)}, \cdots, \boldsymbol{h}_n^{(L)}, \boldsymbol{h}_R\right], \quad \boldsymbol{v} = W^V\left[\boldsymbol{h}_1^{(L)}, \cdots, \boldsymbol{h}_n^{(L)}, \boldsymbol{h}_R\right], \\ \boldsymbol{q}_c = \text{softmax}\left(\frac{\boldsymbol{q}^\intercal\boldsymbol{k}}{\sqrt{d_k}}\right)\boldsymbol{v}.\end{gathered} \tag{4}$$

For LCH-Regret, the final probability $\pi_\theta(a_t|\boldsymbol{x}_t, c)$ is computed as follows:

$$\begin{aligned} u_{\text{Regret}} &= \quad C \cdot \tanh\left(\frac{\boldsymbol{q}_c^\intercal\boldsymbol{W}^L\boldsymbol{h}_R}{\sqrt{d_k}}\right) \odot M \\ u_j &= \begin{cases} C \cdot \tanh\left(\frac{\boldsymbol{q}_c^\intercal\boldsymbol{W}^L\boldsymbol{h}_j}{\sqrt{d_k}}\right) \odot M & j \notin \boldsymbol{x}_t \\ -\infty & \text{otherwise} \end{cases}, \end{aligned} \tag{5}$$

$$\pi_\theta(a_t|\boldsymbol{x}_t, c) = \frac{\exp(u_{a_t})}{\sum_j \exp(u_j) + \exp(u_{\text{Regret}})}, \tag{6}$$

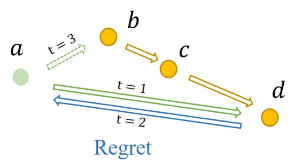
where M is the Regret mask, it will be $-\infty$ if the action is masked and 1 for unmasked actions.

For MatNet-Regret, we use the same way for adding the Regret encoding, except the backbone of which is MatNet [6].

B Details of LCH-Regret

B.1 An Example of the Modified MDP

To better illustrate the design of the modified MDP, we explain the situation provided in Figure 1 in the main text. The corresponding states and actions taken at each time step are listed on the right side of Figure 1.



$s_1 = \{(a), \emptyset\}$	$a_1 = d$
$s_2 = \{(a,d), \emptyset\}$	$a_2 = \text{Regret}$
$s_3 = \{(a), \{(a,d)\}\}$	$a_3 = b$
$s_4 = \{(a,b), \{(a,d)\}\}$	$a_4 = c$
$s_5 = \{(a,b,c), \{(a,d)\}\}$	$a_5 = d$
$s_T = \{(a,b,c,d), \{(a,d)\}\}$	

Figure 1: An example of the modified MDP.

B.2 Regret Mask

The Regret masks of different CO problems are generally designed based on the following criteria.

- Temporarily mask nodes erased by the regret operation of step t at step t+1.
- Mask regret operations in the first two steps of solution generation.
- Mask regret operations in the two steps after each regret is executed.

TSP, and FFSP Mask. For TSP and FFSP, we employ the same Regret mask, which is designed strictly according to the first three criteria mentioned above.

CVRP Mask. Besides the first three criteria, the Regret mask of CVRP employs the following two rules additionally:

- Mask the regret operation in the next two steps when the last node of the partial solution is the depot.
- Mask the regret operation in the **first four steps** of solution generation.

A feasible CVRP solution contains multiple subroutes and an arbitrary permutation of these subroutes represents the same solution $\boldsymbol{x}$. So, rolling back the construction of a new subroute for another subroute is meaningless which may lead to an enlarged action space and an unstable training process. Moreover, we mask the regret operation in the first four construction steps to further ensure a stable training process. Through these rules, we can ensure that regent operations will not result in the same feasible solution $\boldsymbol{x}$ for both TSP and CVRP.

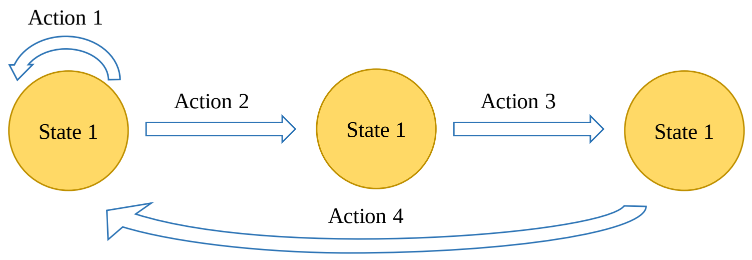


Figure 2: State Transition Graph

Implementation. In practice, we implement a finite state automaton to maintain the Regret mask. For each time step, the finite state automaton will select the regret operation or a node to construct. Simultaneously, the state of the automaton will transition to a new state based on the current state and the selected action. In total, there are 3 states and 4 corresponding actions as follows.

- **State 1:** State 1 represents a normal state when the previous two solution construction steps do not take the regret operation. In State 1, the model can select an unselected node or the regret operation.
- **State 2:** At State 2, as the first and third criterion, the regret operation, the erased node, and all selected nodes are masked (the network operates Regret in the previous step).
- **State 3:** At State 3, as the third criterion, only the regret operation and all selected nodes are masked.
- **Action 1:** Execute when the policy network selects to construct a new node.
- **Action 2:** Execute when the policy network selects the regret operation.
- **Action 3:** Execute Action 3 when the policy network selects to construct a new node and the previous network selection is the regret operation.
- **Action 4:** Execute when the policy network selects to construct a new node and the penultimate last selection is the regret operation.

Figure 2 shows the state transition graph of the finite state automaton. The initial state is State 1 and in the first several steps (2 for FFSP and TSP; 4 for CVRP) the automaton is compulsorily demanded to take action 1 according to the second criterion. Additionally, the Reget mask according to the fourth criterion will be checked and maintained at each time step. The finite state automaton will end after a feasible solution is finally constructed.

C Explanation

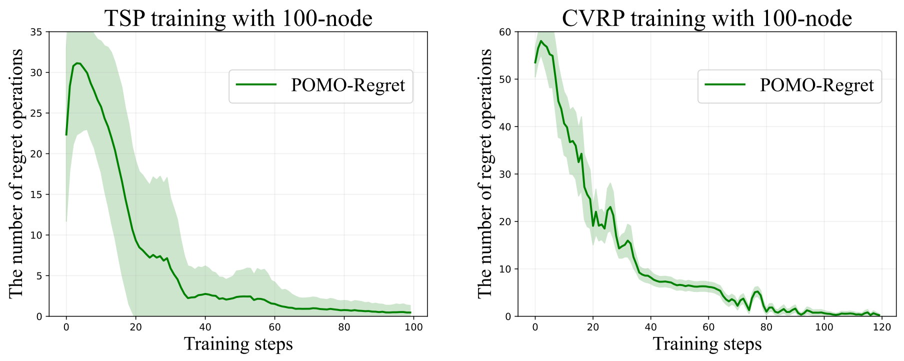
In this part, we prove the theorem below for TSP and CVRP to explain how the proposed Regret encoding $\boldsymbol{h}_{Regret}$ and the additional loss $\mathcal{L}_R$ (Eq. (4) in the main paper) works:

After the Regret encoding is converged to distribution information, $\mathcal{L}_R$ will help to improve the node encoding of less optimal selections in the solution construction process.

Lemma C.1. *As the network converges, the probability of executing Regret operations will decrease.*

Proof. As the policy of LCH-Regret in Eq. (2) in the main part, constructing a feasible complete solution $\boldsymbol{x}$ with Regret archive $\mathbb{R} \neq \emptyset$ will have $\forall\ c,\ p'_\theta(\boldsymbol{x}, \mathbb{R}|c) < p'_\theta(\boldsymbol{x}, \emptyset|c)$ because a softmax probability $\forall\ \boldsymbol{r}\ \forall\ c,\ \pi_\theta(\text{Regret}|\boldsymbol{r}, c) < 1$.

So when trained by the REINFORCE [7] algorithm, if the advantage function $-(f(\boldsymbol{x}) - b(c)) \geq 0$, converged policy network will choose fewer regret operations to maximize probability. If the advantage function $-(f(\boldsymbol{x}) - b(c)) < 0$, the network will learn to decrease $\log \pi_\theta(\text{Regret}|\boldsymbol{r}, c)$. So no matter how $-(f(\boldsymbol{x}) - b(c))$ is signed, $\pi_\theta(\text{Regret}|\boldsymbol{r}, c)$ will decrease. □



(a) TSP (b) CVRP

Figure 3: The number of regret operations.

Figure 3 also proves the lemma by plotting the number of regret operations. The lower number represents a lower probability of Regret operation. For TSP and CVRP problems, the number of regret operations decreases and converges to a low level.

Lemma C.2. *The Regret encoding in a convergent network represents the information of the training distribution.*

Proof. The Regret encoding is a 128-dim parameter with no relation to node encodings so for different instances c sampled from the training distribution, the Regret encoding remains the same. As the lemma C.1, for a converged network, the probability of Regret operations $\forall\ \boldsymbol{r}\ \forall\ c,\ \pi_\theta(\text{Regret}|\boldsymbol{r}, c)$ is low. So because the calculation of the attention decoder is related to the vector distance, the Regret encoding cannot be related to any node encoding, it can only reflect the training distribution information. □

PCA results in Figure 2 in the main body and Appendix D.8 also demonstrate that the PCA result of Regret encoding is often located at the center of node encoding, which showcases that it is not biased toward any nodes or positions.

Lemma C.3. *For a converged LCH-Reget network, if the Regret archive $\mathbb{R} \neq \emptyset$, the advantage function* $-\mathbb{E}_{\boldsymbol{x} \sim p'_\theta(\cdot,\mathbb{R}|c)}\Big[(f(\boldsymbol{x}) - b(c))\Big] > 0$

Proof. A converged LCH-Regret network constructs solutions according to $\pi_\theta(a_t|\boldsymbol{x}_t, c)$ where $\boldsymbol{x}_t$ represent the partial solution in timestep t. If the network chooses the regret operation for the following selection x_{t+1}, the edge (x_{t+1}, x) will not contain in the solution $\boldsymbol{x}$, because the Regret mask masks the Regret operations when they have no impact on the final solution $\boldsymbol{x}$. So, as the network capacity improves to $\pi_\theta(x_{t+1}|\boldsymbol{x}_t, c) \approx \pi^*(a_{t+1}|s_t)$ (the optimal policy in a MDP), the regret operations $\mathbb{R} \neq \emptyset$ harm the performance with $-\mathbb{E}_{\boldsymbol{x} \sim p'_\theta(\cdot,\mathbb{R}|c)}\Big[(f(\boldsymbol{x}) - b(c))\Big] > 0$.

□

Theorem C.4. *When the Regret encoding is converged to distribution information, $\mathcal{L}_R$ helps improve the node encoding which is less optimal in the solution construction process.*

Proof. Similar to the Eq. 6, the attention mechanisms of original LCH methods compute logits as $\boldsymbol{q}_c^\intercal \boldsymbol{k}_i, i \in \{1, ..., n\}$ to decode between n nodes. The $\boldsymbol{q}_c$ is the context embedding which represents the information of the partial solution $\boldsymbol{x}_t$ and $\boldsymbol{k}_i = W^L \boldsymbol{h}_i$ represent the keys in the attention mechanism. And for the proposed LCH-Regret, the log $\pi_\theta(a_t|\boldsymbol{x}_t, c)$ involved in the loss calculation is derived by the equation $\log \pi_\theta(a_t|\boldsymbol{x}_t) = \log \frac{\exp(\boldsymbol{q}_c^\intercal \boldsymbol{k}_{a_t})}{\sum_{i=1}^n \exp(\boldsymbol{q}_t^\intercal \boldsymbol{k}_i) + \exp(\boldsymbol{q}_c^\intercal \boldsymbol{k}_R)}$ where W^L is a linear projection and the $\boldsymbol{k}_R = W^L \boldsymbol{h}_R$ means the key information of the Regret encoding. Moreover, the probability of regret loss $\nabla_\theta \mathcal{L}_R$ has a similar pattern.

$$\begin{aligned}
\nabla_\theta \mathcal{L}_R(c) &= -\mathbb{E}_{\boldsymbol{x},\mathbb{R} \sim p'_\theta(\cdot|c)}\Big[(f(\boldsymbol{x}) - b(c)) \sum_{\boldsymbol{r} \in \mathbb{R}} \nabla_\theta \log \pi_\theta(\text{Regret}|\boldsymbol{r}, c)\Big], \\
&= -\mathbb{E}_{\boldsymbol{x},\mathbb{R} \sim p'_\theta(\cdot|c)}\Big[(f(\boldsymbol{x}) - b(c)) \sum_{\boldsymbol{r} \in \mathbb{R}} \nabla_\theta \log \frac{\exp(\boldsymbol{q}_c^\intercal \boldsymbol{k}_R)}{\sum_{i=1}^n \exp(\boldsymbol{q}_c^\intercal \boldsymbol{k}_i) + \exp(\boldsymbol{q}_c^\intercal \boldsymbol{k}_R)}\Big].
\end{aligned} \tag{7}$$

According to lemma C.3, if we take $f(\boldsymbol{x}) - b(c) < 0$, the loss correlates to the famous InfoNCE loss[8, 9, 10] where the $\boldsymbol{k}_R$ is a negative sample and $k_i, i \in \{1, \cdots, n\}$ are positive samples.

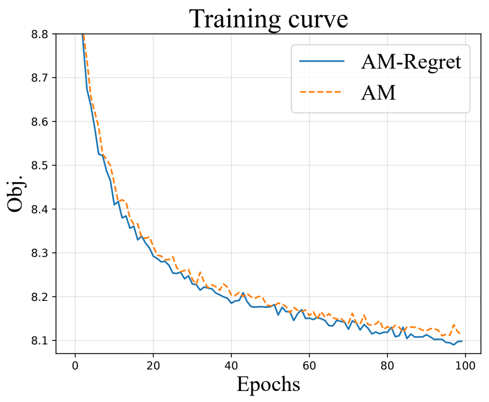
□

In that case, if the model is well-trained, the nodes will avoid choosing the regret. But once a node is badly encoded the choosing probability of regret will increase so that the node will actively adapt to a contrastive-like learning procedure and achieve a better encoding with the help of $\nabla_\theta \mathcal{L}_R$ (when $\mathbb{R} \neq \emptyset$).

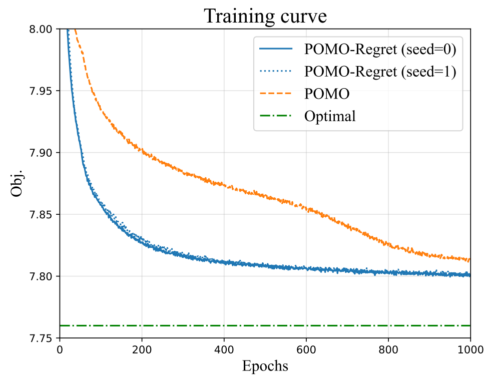
D More Experiments

D.1 Training and Testing

Figure 4 (a) shows the AM-Regret training curve of TSP with 100-node. Due to the typography difficulty, we put it here. Its performance is outstanding as well.



(a) AM-Regret TSP100



(b) POMO-Regret TSP100

Figure 4: Supplementary figures for training curves.

Figure 4 (b) demonstrates the stability of the POMO-Regret on TSP with 100-node. The proposed POMO-Regret has similar performances under different random seeds.

According to the lemma C.3, disabling the regret operation can improve performance. So, LCH-Regret methods disable the regret operation in testing for CVRP and TSP and achieve better solutions.

The baselines involved include Concorde [11][3], HGS [12][4], LKH [13][5], OR tools[6], Attention Model [1][7], RL-CSL [14][8], POMO [2][9], and Sym-NCO [15][10].

On TSP, CVRP, and FFSP, our POMO-Regret trained the same epochs with POMO [2] and AM-Regret trained the same epochs with AM [1] on TSP and CVRP as well. The models used for comparison are the provided pre-trained models for learning-based baselines.

For **LKH3**, we adopt a widely used setup (implemented in the source code of AM [16]), which has no time limit but sets the maximum number of trials in each run to 10K and the total number of runs to 10. With this setup, it takes approximately 6 minutes to solve all 10K TSP50 instances.

In the experiments of FFSP, we follow the settings from Kwon et al. [6][11]. We use the provided test set for experiments, so we directly reported its results in Kwon et al. [6]. In the presentation of comparative algorithms, we adopted the greedy approach of prioritizing the shortest task, as well as the widely used metaheuristics of Genetic Algorithm and Particle Swarm Optimization (referred to as Particle Swarm Opt. in the table).

D.2 Ablation on Learning Regret

To verify the effectiveness of learning the regret mechanism, we have trained a model with a "random" regret mechanism on TSP50 where the model randomly conducts the regret operation with

[3]Concorde: `https://github.com/jvkersch/pyconcorde`
[4]HGS: `https://github.com/vidalt/HGS-CVRP`
[5]LKH: `https://github.com/wouterkool/dpdp`
[6]OR tools: `https://developers.google.com/optimization`
[7]Attention Model: `https://github.com/wouterkool/attention-learn-to-route`
[8]RL-CSL: `https://github.com/zjyuan1208/CPC_AM_master`
[9]POMO: `https://github.com/yd-kwon/POMO/tree/master/NEW_py_ver`
[10]Sym-NCO: `https://github.com/alstn12088/Sym-NCO`
[11]MatNet: `https://github.com/yd-kwon/MatNet`

a probability of $\frac{1}{n}$ at each construction step and the regret encoding remains untrained during training. During inference, the regret operation is disabled like the same setting in POMO-Regret. The results in the table below indicate that the random regret variant is even worse than the original POMO and demonstrate the role of learning the regret mechanism as well.

Gap on TSP50		
POMO	POMO-Regret	Random-Regret (prob=$\frac{1}{50}$)
0.1213%	0.1177%	0.1687%

D.3 More Zero-shot Generalization Tests

To further demonstrate the cross-distribution generalization ability of the proposed POMO-regret method over the baselines, We test the Rotation distribution and Explosion distribution provided in Zhou et al. [16]. The result is shown in the table below. POMO-Regret achieves consistent advantages in these cross-distribution generalization tests.

Table 1: Performance comparison in rotation distribution and explosion distribution.

	TSP200		TSP300		TSP500	
Method	Rotation	Explosion	Rotation	Explosion	Rotation	Explosion
Concord	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
LKH	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
POMO	2.56%	2.35%	9.40%	9.07%	24.21%	24.55%
Sym-NCO	2.56%	1.98%	8.77%	7.59%	22.01%	20.46%
POMO-Regret	**2.20%**	**1.61%**	**7.25%**	**6.65%**	**17.59%**	**17.22%**

D.4 Combination with SGBS and EAS:

This section verifies the performance of the pre-trained model trained with POMO-Regret on large-scale problems. We use SGBS [17] and SGBS+EAS [18] as transfer learning algorithms and test their performance on TSP100, TSP200, and TSP500. The parameters for both algorithms are from the code available at https://github.com/yd-kwon/SGBS. The experimental results are shown in the table below. After incorporating SGBS and SGBS+EAS algorithms, using the proposed POMO-Regret as a pre-trained model can significantly improve its performance compared to using POMO as a pre-trained model.

	TSP100		TSP200		TSP500	
Methods	Obj	Gap	Obj	Gap	Obj	Gap
Concorde	7.7609	-	10.7226	-	16.5153	-
POMO	7.7897	0.37%	10.9296	1.93%	20.2779	22.78%
POMO+SBGS	7.7654	0.06%	10.7931	0.66%	18.3792	11.29%
POMO+SBGS+EAS	7.7637	0.04%	10.7742	0.48%	18.2041	10.23%
POMO-Regret	7.7697	0.11%	10.9050	1.70%	19.8662	20.29%
POMO-Regret+SBGS	7.7646	0.05%	10.7806	0.54%	**17.8612**	**8.15%**
POMO-Regret+SBGS+EAS	**7.7635**	**0.03%**	**10.7655**	**0.40%**	18.0071	9.03%

D.5 Computation Complexity in Training and Testing

Training time is also an important indicator of training-based methods.

The regret mechanism requires additional environment calculations so that LCH-Regret will slightly increase the training time compared to the original model. Theoretically, the time complexity of training POMO on TSP is $\mathcal{O}(n^3d)$ where n is the problem scale, and d is the embedding dimension.

POMO-Regret improves the theoretical complexity to $\mathcal{O}((n + 2r)n^2d)$, where r represents the number of the regret operation. In experiments, the average epoch training time of models is listed as follows. Therefore, the difficulty in training the model will not significantly increase.

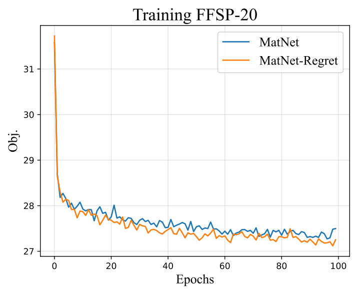
	TSP100	CVRP100	FFSP100
Origin Model	7.2m	1.0m	5.4m
LCH-Regret	11.4m	1.4m	6.3m

Moreover, we have conducted additional tests on large-scale TSP like TSP10000, to validate the scaling ability of POMO and POMO-Regret. As shown in the table below, with the scale increasing, the testing time and the peak memory usage will not significantly improve. So the proposed LCH-Regret is scalable in testing and the proposed POMO-Regret has a similar complexity to POMO when testing.

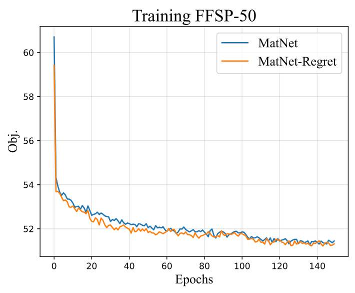
	Testing Time				Peak Memory Usage (MB)			
Problem	TSP50	TSP100	TSP1k	TSP10k	TSP50	TSP100	TSP1k	TSP10k
Instance number	1000	1000	128	1	1000	1000	128	1
POMO	0.17s	0.79s	59.77s	520.24s	555.69	1623.99	13939.42	10355.23
POMO-Regret	0.22s	0.84s	62.24s	529.76s	565.37	1640.13	13957.80	10373.01

D.6 LCH-Regret for FFSP

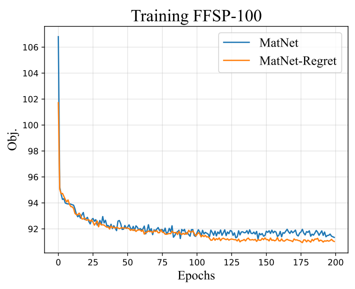
The successful application on FFSP 20, FFSP 50, and FFSP 100 provides further demonstration that the proposed LCH-Regret is a model-agnostic plug-in method. In this section, as a supplement to the content of the paper, we present the convergence curve of MatNet-Regret, which demonstrates that LCH-Regret effectively improves the convergence speed.



(a) FFSP20



(b) FFSP50



(c) FFSP100

Figure 5: Training curves on FFSP.

D.7 TSPLIB

This subsection gives evaluation performances over methods in the TSPLIB ($100 \leq N < 300$). Models are trained on TSP with 100-node datasets. Objective values (named Obj.) and Optimal Gaps (named Gap) are shown in Table 2. In our implementation, for consistency to the default in Kim et al. [15], all three methods are without 8 augments. As a model-agnostic plug-in method, LCH-Regret can significantly improve the performance of POMO and Sym-NCO on TSPLIB.

Table 2: Performance comparison in real-world instances in TSPLIB.

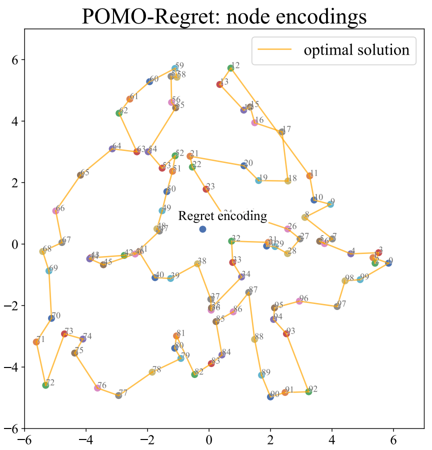
		POMO		Sym-NCO		POMO-Regret		Sym-NCO-Regret	
instance	opt	obj	Gap	obj	Gap	obj	Gap	obj	Gap
a280	2579	2908	12.76%	2950	14.39%	2927	13.49%	**2881**	11.71%
bier127	118282	134512	13.72%	125496	6.10%	128145	8.34%	**124434**	5.20%
ch130	6110	6130	0.33%	6140	0.49%	6130	0.33%	6130	0.33%
ch150	6528	6566	0.58%	6570	0.64%	6574	0.70%	**6562**	0.52%
d198	15780	20359	29.02%	**19128**	21.22%	20486	29.82%	21272	34.80%
eil101	629	641	1.91%	641	1.91%	641	1.91%	641	1.91%
gil262	2378	2514	5.72%	2478	4.21%	2481	4.33%	**2474**	4.04%
kroA100	21282	21606	1.52%	21683	1.88%	21714	2.03%	**21583**	1.41%
kroB100	22141	22372	1.04%	22917	3.50%	22367	1.02%	**22321**	0.81%
kroC100	20749	20894	0.70%	21282	2.57%	21299	2.65%	**20788**	0.19%
kroD100	21294	21819	2.47%	22039	3.50%	**21844**	2.58%	21852	2.62%
kroE100	22068	22289	1.00%	22313	1.11%	22458	1.77%	**22284**	0.98%
kroA150	26524	**27109**	2.21%	27185	2.49%	27159	2.39%	27285	2.87%
kroB150	26130	26969	3.21%	27185	4.04%	**26620**	1.88%	26840	2.72%
kroA200	29368	30365	3.39%	30427	3.61%	30414	3.56%	**30007**	2.18%
kroB200	29437	30889	4.93%	30892	4.94%	30435	3.39%	**30355**	3.12%
lin105	14379	**14454**	0.52%	14909	3.69%	14754	2.61%	14508	0.90%
pr107	44303	44980	1.53%	**44729**	0.96%	45333	2.32%	45321	2.30%
pr124	59030	59544	0.87%	59838	1.37%	**59187**	0.27%	59850	1.39%
pr136	96772	97895	1.16%	**97697**	0.96%	97870	1.13%	97707	0.97%
pr144	58537	59932	2.38%	58935	0.68%	58748	0.36%	**58672**	0.23%
pr152	73682	75049	1.86%	76078	3.25%	**74412**	0.99%	75170	2.02%
pr226	80369	83622	4.05%	83093	3.39%	83621	4.05%	**82440**	2.58%
pr264	49135	55942	13.85%	54753	11.43%	55708	13.38%	**54260**	10.43%
pr299	48191	56771	17.80%	55058	14.25%	54009	12.07%	**53970**	11.99%
rat195	2323	2546	9.60%	2563	10.33%	**2496**	7.45%	2532	9.00%
rd100	7910	7924	0.18%	7924	0.18%	7919	0.11%	**7915**	0.06%
ts225	126643	132623	4.72%	**128349**	1.35%	128780	1.69%	130085	2.72%
tsp225	3916	4183	6.82%	**4122**	5.26%	4126	5.36%	4175	6.61%
u159	42080	**42480**	0.95%	42566	1.15%	42754	1.60%	42519	1.04%
Overall			5.03%		4.49%		4.45%		4.25%

D.8 More Encodings

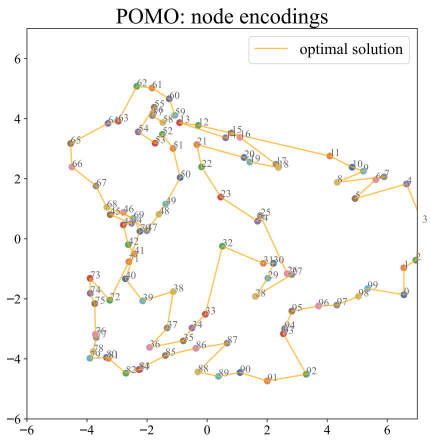
In this part, we provide more evidence for the conclusion that LCH-Regret achieves better encodings.

For TSP instances, Figure 6 shows the distribution of encodings after 500 trained epochs (the same as Figure 5 in the main part), while Figure 7 shows the PCA encodings distribution of converged model used in the baseline comparison.

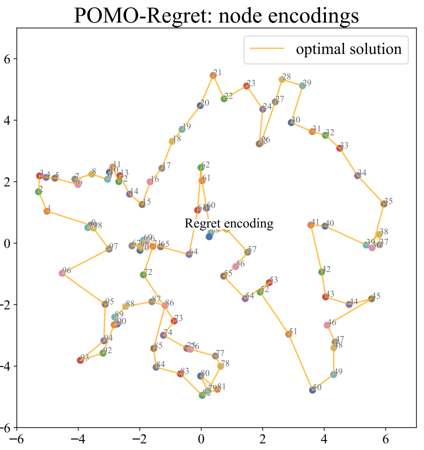
What's more, for CVRP, Figure 8 instantiates that node encodings achieved by POMO-Regret also have a better distribution.



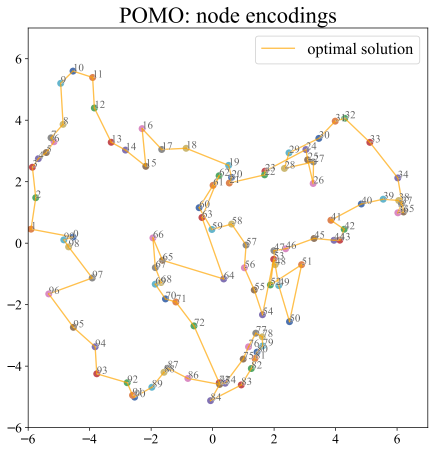
(a) POMO-Regret (Ours) 500 epoches



(b) POMO [2] 500 epoches

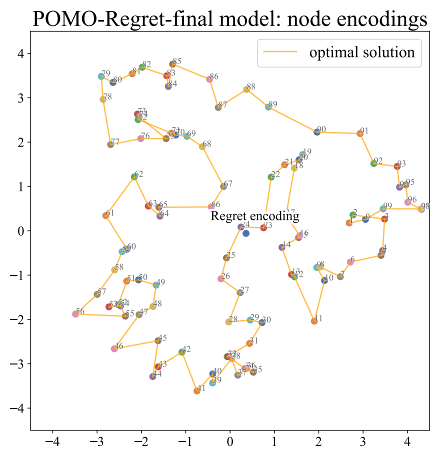


(c) POMO-Regret (Ours) 500 epoches

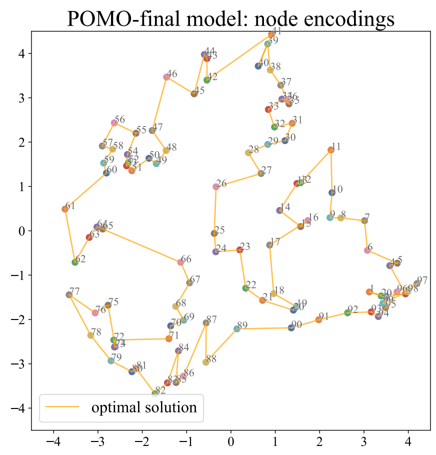


(d) POMO [2] 500 epoches

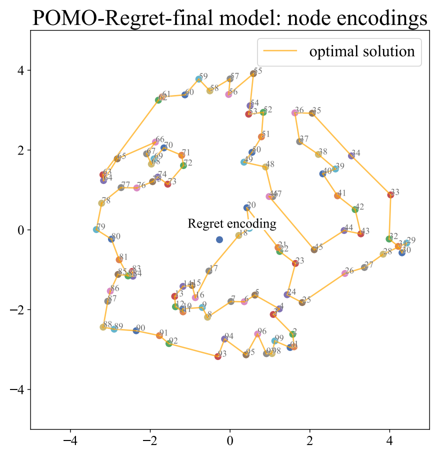
Figure 6: Principal component analysis (PCA) [19] of TSP encodings. Compared to POMO [2], after the same training steps (**500 epoches**), our POMO-Regret achieves more uniform and sparse encodings.



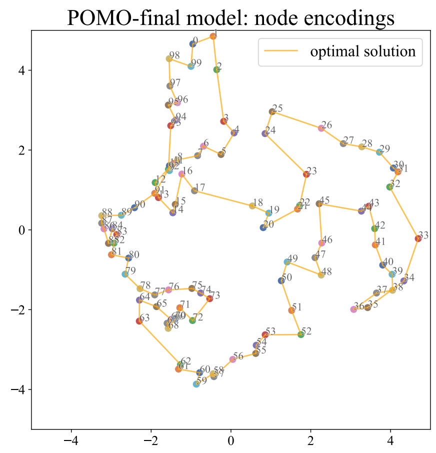
(a) POMO-Regret (Ours) 3100 epoches



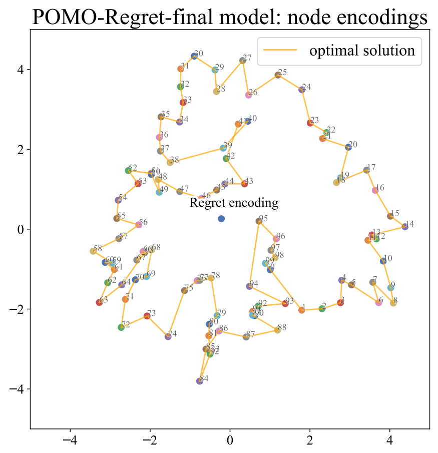
(b) POMO [2] 3100 epoches



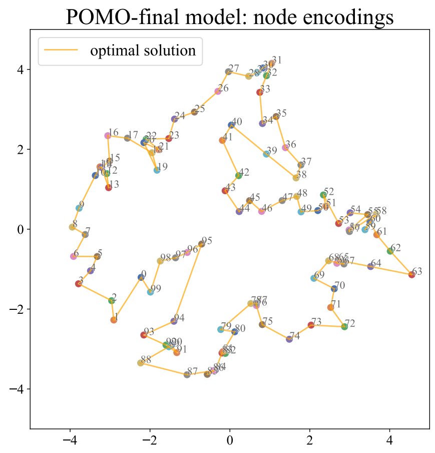
(c) POMO-Regret (Ours) 3100 epoches



(d) POMO [2] 3100 epoches

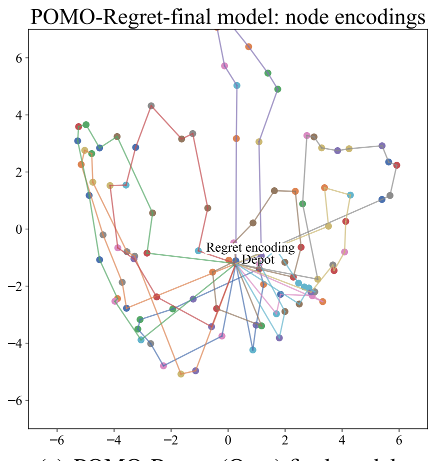


(e) POMO-Regret (Ours) 3100 epoches

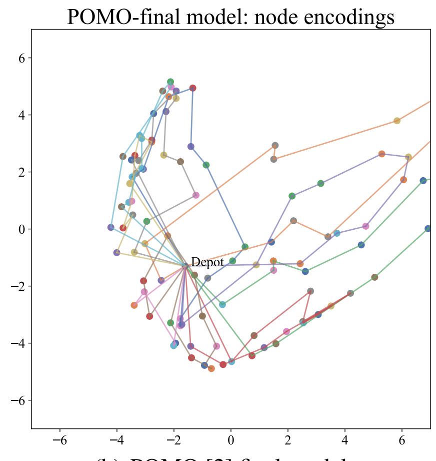


(f) POMO [2] 3100 epoches

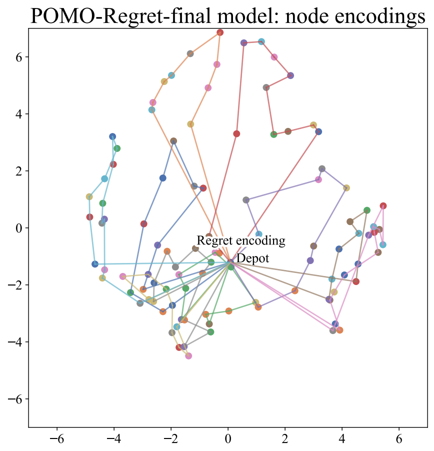
Figure 7: Principal component analysis (PCA) [19] of TSP encodings. Compared to POMO [2], after the same training steps (**3100 epoches**), our POMO-Regret achieves more uniform and sparse encodings.



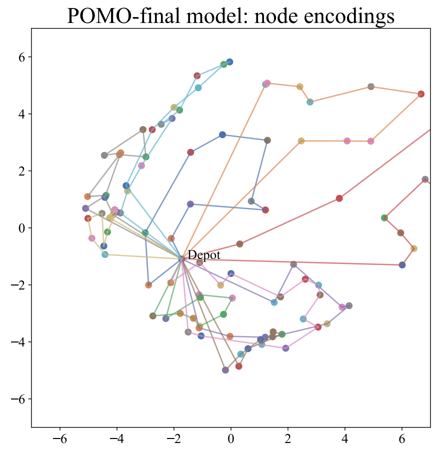
(a) POMO-Regret (Ours) final model



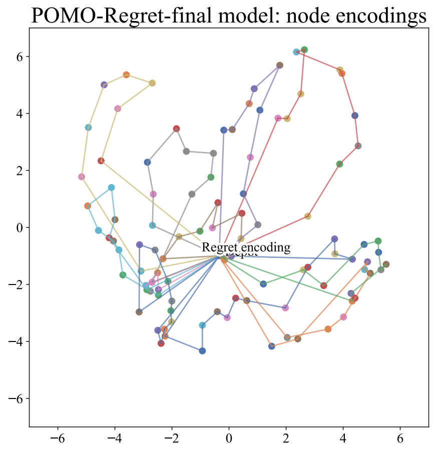
(b) POMO [2] final model



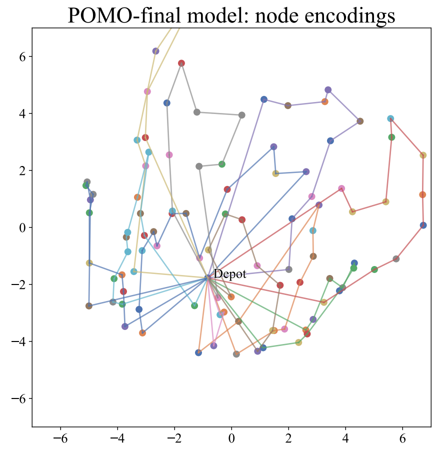
(c) POMO-Regret (Ours) final model



(d) POMO [2] final model



(e) POMO-Regret (Ours) final model



(f) POMO [2] final model

Figure 8: Principal component analysis (PCA) [19] of CVRP encodings. Compared to POMO [2], after the same training steps (**8100 epoches**), our POMO-Regret achieves more uniform and sparse encodings.

D.9 License

Table 3: A summary of licenses.

Resources	Type	License
HGS	Code	MIT License
Concrde	Code	BSD 3-Clause License
OR-Tools	Code	Apache License, Version 2.0
LKH3	Code	Available for academic research use
AM	Code	MIT License
RL-CSL	Code	MIT License
POMO	Code	MIT License
Sym-NCO	Code	MIT License
MatNet	Code	MIT License
TSPLIB	Dataset	Available for any non-commercial use

The licenses for the code and dataset used in this work are listed in Table 3.

References

[1] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019.

[2] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf`.

[3] 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. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[5] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[6] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5138–5149. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/29539ed932d32f1c56324cded92c07c2-Paper.pdf`.

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

[8] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[9] Laurence Aitchison. Infonce is a variational autoencoder. *arXiv preprint arXiv:2107.02495*, 2021.

[10] Chuhan Wu, Fangzhao Wu, and Yongfeng Huang. Rethinking infonce: How many negative samples do you need? *arXiv preprint arXiv:2105.13003*, 2021.

[11] David Applegate, Robert Bixby, V Chvatal, and William. Cook. Concorde tsp solver. `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006. Accessed: 2023-12-19.

[12] Thibaut Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[13] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[14] Zhongju Yuan, Genghui Li, Zhenkun Wang, Jianyong Sun, and Ran Cheng. Rl-csl: A combinatorial optimization method using reinforcement learning and contrastive self-supervised learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):1010–1024, 2023. doi: 10.1109/TETCI.2021.3139802.

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

[16] Jianan Zhou, Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Towards omni-generalizable neural methods for vehicle routing problems, 2023.

[17] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. Simulation-guided beam search for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:8760–8772, 2022.

[18] André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*, 2021.

[19] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.