Supplementary Materials of Submission: Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances

Anonymous Authors[1]

1. Architecture of the Att-GCRN model

As shown in Fig. 1, the Att-GCRN model uses three steps to convert the input information (the coordinates of each vertex, as well as the length of each edge) into a heat map, i.e., the probability of each edge belonging to the optimal solution. At first, it embeds the input information into a series of vectors (each vertex or each edge corresponds to a 300×1 vector). Then, a feature extractor is used to extract the weights information of the vertices and edges. Finally, the information is fed into a classification network to build the heat map.

Additionally, to train the model, the solutions produced by exact solver Concorde are used as the ground-truth solutions, while a focal loss function is chosen as the loss function, and the adaptive moment estimation method (Kingma & Ba, 2014) is used as the gradient descent optimizer (with an initial learning rate of 10^{-2}).

1.1. Graph embedding

Given an undirected graph G with n vertices, we at first embed each vertex and each edge into a vector (Levy & Goldberg, 2014) as follows:

$$\begin{aligned} \boldsymbol{v}_i^0 &= \boldsymbol{W}_1 \boldsymbol{c}_i + \boldsymbol{A}_i, \\ \boldsymbol{e}_{ij}^0 &= \boldsymbol{W}_2 \boldsymbol{d}_{ij} + \boldsymbol{B}_{ij}. \end{aligned} \tag{1}$$

where $\boldsymbol{v}_i^0$ and $\boldsymbol{e}_{ij}^0$ respectively denotes the embedded vector (300×1) corresponding to vertex i and edge (i, j). Respectively, $\boldsymbol{W}_1$ is a 300×2 matrix, $\boldsymbol{c}_i$ denotes the coordinates of vertex i (represented as a 2×1 vector), $\boldsymbol{A}_i$ is a 300×1 vector corresponding to vertex i, $\boldsymbol{W}_2$ is a 300×1 vector, $\boldsymbol{d}_{ij}$ denotes the distance of edge (i, j), $\boldsymbol{B}_{ij}$ is also a 300×1 vector corresponding to edge (i, j). Notice that all the elements of $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{A}_i$ and $\boldsymbol{B}_{ij}$ are trainable parameters, which are randomly initialized, and updated via gradient descent optimizer during the training process.

Overall, the embedding of all the vertices forms a $300 \times n$ matrix $\boldsymbol{v}^0$, and the embedding of all the edges forms a $300 \times n \times n$ tensor $\boldsymbol{e}^0$, which are fed into the feature extractor.

1.2. Feature extractor

The feature extractor consists of L (L=30 in this paper) graph residual convolutional layers. Let $\mathbf{H}^l = \{\boldsymbol{v}^l, \boldsymbol{e}^l\}$ denote the graph feature of the lth layer, whose elements are recursively calculated as follows:

$$\begin{aligned} \boldsymbol{v}_i^l = \boldsymbol{v}_i^{l-1} + \mathrm{ReLU}\Big(\mathrm{BN}\big(\mathrm{Conv1d}(\boldsymbol{v}_i^{l-1}) \\ + \sum_{j \neq i} \alpha_{ij}^l \odot \mathrm{Conv1d}(\boldsymbol{v}_j^{l-1})\big)\Big), \\ \boldsymbol{e}_{ij}^l = \boldsymbol{e}_{ij}^{l-1} + \mathrm{ReLU}\Big(\mathrm{BN}\big(\mathrm{Conv1d}^*(\boldsymbol{e}_{ij}^{l-1}) \\ + \mathrm{Conv1d}(\boldsymbol{v}_i^{l-1}) + \mathrm{Conv1d}(\boldsymbol{v}_j^{l-1})\big)\Big). \end{aligned} \tag{2}$$

where ReLU denotes rectified linear unit, BN denotes batch normalization, $\mathrm{Conv1d}$ denotes a one-dimensional convolutional layer with the stride of 1 and 300 kernels of size 1×1 (Glorot et al., 2011; Ioffe & Szegedy, 2015). $\mathrm{Conv1d}^*$ is similar to $\mathrm{Conv1d}$, but with 300 kernels of size $1 \times 1 \times 1$ (the output of $\mathrm{Conv1d}(\boldsymbol{v}_i^{l-1})$ and $\mathrm{Conv1d}^*(\boldsymbol{e}_{ij}^{l-1})$ are both 300×1 vectors). α_{ij}^l denotes the attention coefficient relative to edge (i, j), which is defined as follows:

$$\alpha_{ij}^l = \frac{\exp\Big(\mathrm{LeakyReLU}\big(\mathrm{Conv1d}^\diamond(\boldsymbol{e}_{ij}^l)\big)\Big)}{\sum_{k \neq i} \exp\Big(\mathrm{LeakyReLU}\big(\mathrm{Conv1d}^\diamond(\boldsymbol{e}_{ik}^l)\big)\Big)}, \tag{3}$$

where $\mathrm{LeakyReLU}$ denotes leaky rectified linear unit, $\mathrm{Conv1d}^\diamond$ is a one-dimensional convolutional layer which only consists of one $1 \times 1 \times 1$ kernel with the stride of 1 (Maas et al., 2013), and $\mathrm{Conv1d}^\diamond(\boldsymbol{e}_{ij}^l)$ outputs a real value.

1.3. Classification network

Based on the information of $\boldsymbol{e}^L$, the classification network tries to build a heat map by feed-forward propagation:

$$\boldsymbol{P} = \mathrm{MLP}(\boldsymbol{e}^L) \tag{4}$$

where MLP denotes a multi-layer perception (Cybenko, 1989) with three full connected layers. $\boldsymbol{P}$ denotes the heat map (a $n \times n$ matrix), whose element $\boldsymbol{P}_{ij}$ estimates the probability of edge (i, j) belonging to the optimal solution.

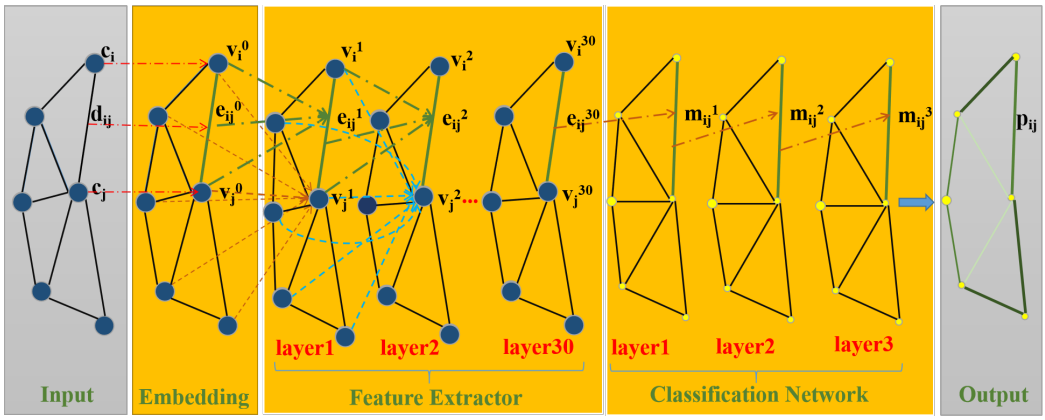


Figure 1. Architecture of the Att-GCRN model.

1.4. Loss function

The model in trained in batch (each batch contains M=500 training instances), and a focal loss function (Lin et al., 2017) is used to calculate the accumulated loss:

$$\mathrm{FL}(\boldsymbol{P}) = -\frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\Big(\beta_0(1-\boldsymbol{I}_{ij}^{(m)}) + \beta_1 \times \mathbf{I}_{ij}^{(m)}\Big)\Big(\mathbf{I}_{ij}^{(m)} - \boldsymbol{P}_{ij}^{(m)}\Big)\log(\boldsymbol{P}_{ij}^{(m)}). \tag{5}$$

$\boldsymbol{I}^{(m)}$ is a $n \times n$ matrix corresponding the mth training instance, whose element $\boldsymbol{I}_{ij}^{(m)} = 1$ if edge (i, j) belongs to the ground-truth solution of the m-th training instance, and $\boldsymbol{I}_{ij}^{(m)} = 0$ otherwise. $\boldsymbol{P}_{ij}^{(m)}$ is an element of the m-th heat map (built by the classification network). β_0 and β_1 are balanced coefficients relative to the edges not belonging to (or belonging to) the ground-truth solutions, which are calculated as follows (Liu & Zhou, 2006):

$$\beta_0 = \frac{n^2}{n^2 - 2n},\ \beta_1 = \frac{n^2}{2n}. \tag{6}$$

1.5. Additional information

Training platform: one GTX 1080 Ti GPU.

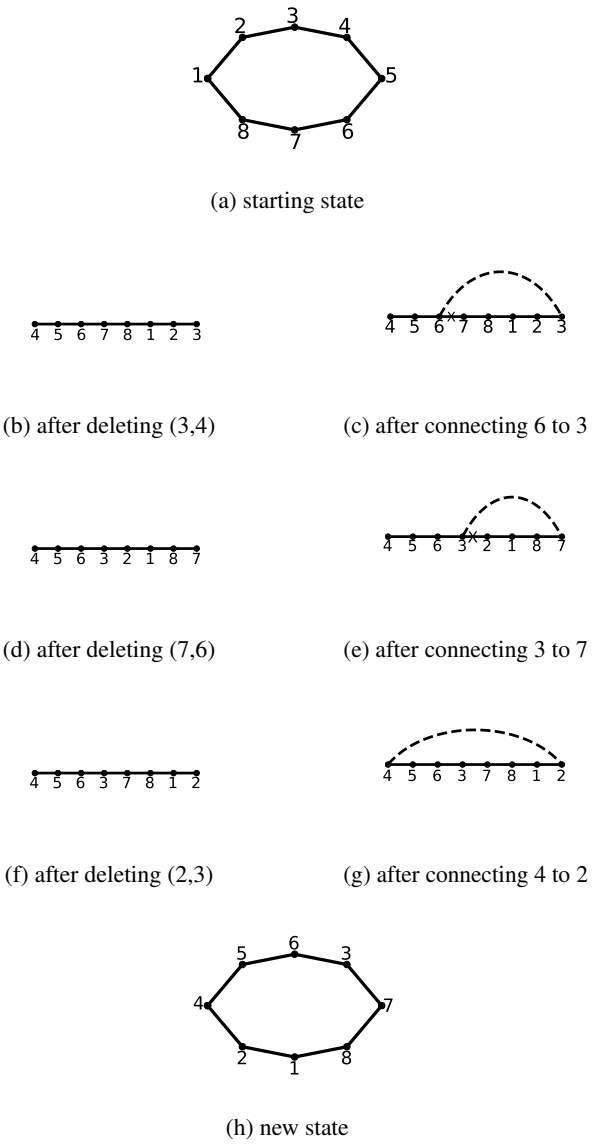
Training set: we train two models, i.e., TSP20-model and TSP50-model (respectively, each input graph has 20 or 50 vertices), which are respectively used to tackle small-scale (with $n \leq 100$) or large-scale (with $n > 100$) instances. Respectively, we use the 990,000 instances with 20 vertices (50 vertices) generated in (Joshi et al., 2019) to train our TSP20-model (TSP50-model).

Training time: 12 hours for TSP20-model (970 epoches), 25 hours for TSP50-model (1270 epoches), excluding the time elapsed by Concorde to produce ground-truth solutions.

2. Example of applying an action during MDP

As described previously, during the Markov decision process (MDP), each action is represented as $\boldsymbol{a} = (a_1, b_1, a_2, b_2, \ldots, a_k, b_k, a_{k+1})$, where k is a variable and the final vertex a_{k+1} must coincide with a_1.

Fig.2 exemplifies the process of applying an action. Respectively, sub-figure (a) is the starting state $\boldsymbol{\pi} = (1, 2, 3, 4, 5, 6, 7, 8)$. To determine an action, we at first decide a vertex a_1 and delete the edge between a_1 and its previous vertex b_1. Without loss of generality, suppose $a_1 = 4$, then $b_1 = 3$ and edge $(3, 4)$ is deleted, resulting in a temporary status shown in sub-figure (b) (for the sake of clarity, drawn as a line which starts from a_1 and ends at b_1). Furthermore, we decide a vertex a_2 to connect vertex b_1, generally resulting in an unfeasible solution containing an inner cycle (unless $a_2 = a_1$). For example, suppose $a_2 = 6$ and connect it to vertex 3, the resulting temporary status is shown in sub-figure (c), where an inner cycle occurs and the degree of vertex a_2 increases to 3. To break inner cycle and reduce the degree of a_2 to 2, the edge between vertex a_2 and vertex $b_2 = 7$ should be deleted, resulting in a temporary status shown in sub-figure (d). This process is repeated, to get a series of vertices a_k and b_k ($k \geq 2$). In this example, $a_3 = 3$ and $b_3 = 2$, respectively corresponding to sub-figures (e) and (f). Once $a_{k+1} = a_1$, the loop closes and reaches a new state (feasible TSP solution). For example, if let $a_4 = a_1 = 4$ and connect a_4 to b_3, the resulting new state is shown in sub-figure (g), which is redrawn as a cycle in sub-figure (h).



(a) starting state

(b) after deleting (3,4)

(c) after connecting 6 to 3

(d) after deleting (7,6)

(e) after connecting 3 to 7

(f) after deleting (2,3)

(g) after connecting 4 to 2

(h) new state

Figure 2. The decision process of an example action

3. Optimality of the exact solver Concorde

Although Concorde is an exact solver, however, on some instances, the solutions produced by Concorde are not strictly optimal (we do not know the reason), which could still be slightly improved by our algorithm. For example, for the first instance with 20 vertices, the solution produced by Concorde is (1,8,19,7,11,4,13,20,10,5,15,9,18,**3,6,12**,14,2,16,17,1), corresponding to a total distance of **3.347795**. For comparison, our algorithm produces a different solution (1,8,19,7,11,4,13,20,10,5,15,9,18,**3,12,6**,14,2,16,17,1) with different permutations (**bold** ones), corresponding to a total distance of **3.347631**, better than Concorde. There are many similar instances among the data sets used in this paper.

4. Extensive experiments on large instances with 10,000 vertices

Additionally, in order to further verify the generalization ability of our algorithm, we similarly generate 16 large instances, each with 10,000 vertices. On these large instances, several baseline algorithms face a big challenge. For example, the three learning based algorithms proposed in (Joshi et al., 2019) all fail due to memory exception (tested on the same platform as previously described), while the two exact solvers (Concorde and Gurobi) as well as the two GAT models in (Deudon et al., 2018) all fail due to time exception (up to five hours is allowed for each instance). Therefore, we exclude these seven baseline algorithms, and just compare our Att-GCRN+MCTS algorithm and the remaining three learning based algorithms (Kool et al., 2019), all evaluated on one GTX 1080 Ti GPU. The results produced by LKH3 (evaluated on one Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz) are listed for indicative purpose.

As shown in Table 1, Att-GCRN+MCTS is able to produce solutions close to LKH3, corresponding to a small average gap of 4.0780%. By contrast, the three learning based algorithms correspond to a huge average gap of 501.2737%, 97.3932%, 80.2802% respectively. The runtime of our algorithm remains reasonable (shorter than the best one of the three baselines). These results confirm that our approach can be smoothly generalized to very large TSP instances.

Table 1. Performance of Att-GCRN+MCTS with respect to five baselines, tested on 16 TSP instances with 10,000 vertices.

Method	Type	TSP10000		
		Length	Gap (vs. LKH3)	Time
LKH3	Heuristic	71.7778	-	8.8h
GAT (Kool et al., 2019)	RL, S	431.5812	501.2737%	12.63m
GAT (Kool et al., 2019)	RL, G	141.6846	97.3932%	5.99m
GAT (Kool et al., 2019)	RL, BS	129.4012	80.2802%	1.81h
Att-GCN+MCTS (**Ours**)	SL+RL	74.7049	**4.0780%**	4.16m + 1.52h

5. Sensitivity analysis of parameters

We randomly select 348 instances (100 instances from TSP20, TSP50, TSP100, and 16 instances from TSP200, TSP500, TSP1000 respectively) as sample instances, based on which we analyze the sensitivity of hyper parameters ω, α, β, H (for parameter T, we set it manually to make sure that the our total runtime remain reasonable). For each parameter, we implement ten scenarios by varying the chosen parameter within a reasonable range, while fixing the other parameters in accordance with the default settings ($\omega = 5$, $\alpha = 1$, $\beta = 10$, $H = 10n$ by default). Then we rerun experiments on the sample instances, and list the results in the following tables, from which we observe that the time needed for generating and merging heat-maps heavily depend on the value of ω, while the algorithm performs quite robustly relative to parameters α, β and H.

Reference

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. Learning heuristics for the tsp by policy gradient. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 170–181. Springer, 2018.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Levy, O. and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pp. 2177–2185, 2014.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Liu, X.-Y. and Zhou, Z.-H. The influence of class imbalance on cost-sensitive learning: An empirical study. In *Sixth International Conference on Data Mining (ICDM'06)*, pp. 970–974. IEEE, 2006.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3, 2013.

Table 2. Results relative to different values of hyper-parameter ω.

Parameter	Metrics	TSP-50	TSP-100	TSP-200	TSP-500	TSP-1000
$\omega = 1$	Gap	0.0838%	0.2963%	52.7880%	93.1805%	95.3237%
	Time	2.48s + 34.85s	3.36s + 68.13s	0.35s + 12.09s	0.84s + 32.45s	1.56s + 64.60s
$\omega = 2$	Gap	0.0016%	0.0010%	0.6482%	5.3072%	10.9565%
	Time	3.69s + 34.64s	11.30s + 68.08s	0.51s + 12.18s	1.10s + 32.14s	2.44s + 63.64s
$\omega = 3$	Gap	-0.0022%	-0.0019%	0.4978%	1.4883%	2.1437%
	Time	5.72s + 34.80s	16.43s + 68.67s	0.67s + 12.88s	1.53s + 32.08s	3.23s + 63.03s
$\omega = 4$	Gap	-0.0030%	-0.0022%	0.5652%	1.6978%	2.3156%
	Time	6.99s + 34.02s	22.41s + 68.02s	0.81s + 12.22s	1.91s + 32.19s	4.25s + 64.54s
$\omega = 5$	Gap	-0.0036%	-0.0024%	0.3043%	1.4050%	1.8280%
	Time	15.54s + 34.19s	23.64s + 68.61s	2.58s + 12.77s	3.89s + 32.61s	5.49s + 63.56s
$\omega = 8$	Gap	-0.0035%	-0.0015%	0.6270%	1.7718%	2.6302%
	Time	13.98s + 34.22s	42.85s + 68.33s	1.52s + 12.39s	3.71s + 32.95s	7.63s + 63.83s
$\omega = 10$	Gap	-0.0035%	-0.0016%	0.5993%	1.7543%	2.8248%
	Time	17.25s + 34.14s	54.54s + 68.93s	1.76s + 12.88s	4.48s + 32.03s	9.38s + 63.51s
$\omega = 20$	Gap	-0.0035%	-0.0011%	0.5190%	1.8004%	3.0494%
	Time	33.97s + 34.14s	108.11s + 68.93s	3.47s + 12.88s	9.46s + 32.03s	18.70s + 63.51s
$\omega = 30$	Gap	-0.0037%	-0.0006%	0.6399%	1.8202%	3.0653%
	Time	50.50s + 34.36s	179.05s + 68.74s	4.99s + 12.96s	15.89s + 32.50s	26.66s + 64.45s
$\omega = 50$	Gap	-0.0037%	-0.0015%	0.8036%	1.8068%	3.2864%
	Time	84.31s + 34.22s	262.55s + 68.58s	8.57s + 12.81s	38.88s + 32.66s	43.96s + 63.54s

Table 3. Results relative to different values of hyper-parameter α.

Parameter	Metrics	TSP-20	TSP-50	TSP-100	TSP-200	TSP-500	TSP-1000
$\alpha = 0.2$	Gap	-0.0001%	-0.0033%	-0.0022%	0.4488%	1.5984%	1.9417%
	Time	2.33s + 14.12s	15.54s + 34.19s	23.64s + 68.76s	2.58s + 12.26s	3.89s + 32.77s	5.49s + 64.58s
$\alpha = 0.4$	Gap	-0.0001%	-0.0033%	-0.0023%	0.3423%	1.5124%	1.9716%
	Time	2.33s + 14.11s	15.54s + 34.14s	23.64s + 68.45s	2.58s + 12.19s	3.89s + 33.20s	5.49s + 63.89s
$\alpha = 0.6$	Gap	-0.0001%	-0.0036%	-0.0022%	0.3284%	1.5709%	1.8754%
	Time	2.33s + 14.14s	15.54s + 34.09s	23.64s + 68.54s	2.58s + 12.24s	3.89s + 32.51s	5.49s + 63.21s
$\alpha = 0.8$	Gap	-0.0001%	-0.0034%	-0.0022%	0.3006%	1.4713%	1.9525%
	Time	2.33s + 14.11s	15.54s + 34.20s	23.64s + 68.75s	2.58s + 12.30s	3.89s + 32.64s	5.49s + 63.68s
$\alpha = 1$	Gap	-0.0001%	-0.0036%	-0.0024%	0.3043%	1.4050%	1.8280%
	Time	2.33s + 14.13s	15.54s + 34.29s	23.64s + 68.91s	2.58s + 12.56s	3.89s + 33.79s	5.49s + 66.29s
$\alpha = 2$	Gap	-0.0001%	-0.0034%	-0.0026%	0.3664%	1.5329%	1.9169%
	Time	2.33s + 14.11s	15.54s + 34.16s	23.64s + 68.58s	2.58s + 12.25s	3.89s + 33.65s	5.49s + 66.47s
$\alpha = 4$	Gap	-0.0001%	-0.0034%	-0.0023%	0.4636%	1.6604%	1.8405%
	Time	2.33s + 14.11s	15.54s + 34.14s	23.64s + 68.68s	2.58s + 12.28s	3.89s + 34.15s	5.49s + 63.57s
$\alpha = 6$	Gap	-0.0001%	-0.0035%	-0.0022%	0.4654%	1.5744%	1.9206%
	Time	2.33s + 14.11s	15.54s + 34.12s	23.64s + 68.60s	2.58s + 12.27s	3.89s + 33.65s	5.49s + 65.24s
$\alpha = 8$	Gap	-0.0001%	-0.0033%	-0.0022%	0.4636%	1.5926%	1.9363%
	Time	2.33s + 14.12s	15.54s + 34.14s	23.64s + 68.69s	2.58s + 12.44s	3.89s + 34.23s	5.49s + 66.47s
$\alpha = 10$	Gap	-0.0001%	-0.0034%	-0.0017%	0.4802%	1.4637%	1.9181%
	Time	2.33s + 14.11s	15.54s + 34.58s	23.64s + 69.19s	2.58s + 12.42s	3.89s + 32.17s	5.49s + 64.13s

Table 4. Results relative to different values of hyper-parameter β.

Parameter	Metrics	TSP-20	TSP-50	TSP-100	TSP-200	TSP-500	TSP-1000
$\beta = 1$	Gap	-0.0001%	-0.0034%	-0.0018%	0.5781%	1.6095%	1.9927%
	Time	2.33s + 14.30s	15.54s + 34.47s	23.64s + 68.80s	2.58s + 12.01s	3.89s + 32.03s	5.49s + 63.74s
$\beta = 2$	Gap	-0.0001%	-0.0033%	-0.0019%	0.5329%	1.5949%	2.0722%
	Time	2.33s + 14.25s	15.54s + 34.61s	23.64s + 68.79s	2.58s + 12.70s	3.89s + 32.22s	5.49s + 64.54s
$\beta = 3$	Gap	-0.0001%	-0.0035%	-0.0019%	0.5357%	1.6802%	2.0192%
	Time	2.33s + 14.27s	15.54s + 34.54s	23.64s + 68.21s	2.58s + 12.13s	3.89s + 32.72s	5.49s + 64.63s
$\beta = 5$	Gap	-0.0001%	-0.0034%	-0.0017%	0.3720%	1.6306%	2.0751%
	Time	2.33s + 14.27s	15.54s + 34.78s	23.64s + 68.94s	2.58s + 12.42s	3.89s + 32.65s	5.49s + 63.76s
$\beta = 8$	Gap	-0.0001%	-0.0034%	-0.0019%	0.5652%	1.6907%	1.9716%
	Time	2.33s + 14.27s	15.54s + 34.31s	23.64s + 68.06s	2.58s + 12.26s	3.89s + 32.07s	5.49s + 63.51s
$\beta = 10$	Gap	-0.0001%	-0.0036%	-0.0024%	0.3043%	1.4050%	1.8280%
	Time	2.33s + 14.13s	15.54s + 34.29s	23.64s + 68.91s	2.58s + 12.56s	3.89s + 33.79s	5.49s + 66.29s
$\beta = 20$	Gap	-0.0001%	-0.0033%	-0.0019%	0.4331%	1.5346%	1.9160%
	Time	2.33s + 14.26s	15.54s + 34.24s	23.64s + 68.86s	2.58s + 12.13s	3.89s + 32.44s	5.49s + 64.12s
$\beta = 30$	Gap	-0.0001%	-0.0034%	-0.0019%	0.5135%	1.5774%	2.0676%
	Time	2.33s + 14.27s	15.54s + 34.85s	23.64s + 68.26s	2.58s + 12.52s	3.89s + 32.19s	5.49s + 64.73s
$\beta = 50$	Gap	-0.0001%	-0.0032%	-0.0021%	0.4580%	1.5680%	2.0788%
	Time	2.33s + 14.27s	15.54s + 34.46s	23.64s + 68.58s	2.58s + 12.15s	3.89s + 32.30s	5.49s + 64.01s
$\beta = 100$	Gap	-0.0001%	-0.0031%	-0.0021%	0.4728%	1.7106%	2.0246%
	Time	2.33s + 14.28s	15.54s + 34.00s	23.64s + 68.22s	2.58s + 12.20s	3.89s + 32.84s	5.49s + 64.07s

Table 5. Results relative to different values of hyper-parameter H.

Parameter	Metrics	TSP-20	TSP-50	TSP-100	TSP-200	TSP-500	TSP-1000
$H = 1n$	Gap	-0.0001%	-0.0034%	-0.0016%	0.6224%	1.7433%	2.2872%
	Time	2.33s + 13.99s	15.54s + 32.49s	23.64s + 68.70s	2.58s + 12.04s	3.89s + 32.30s	5.49s + 64.11s
$H = 2n$	Gap	-0.0001%	-0.0032%	-0.0021%	0.5033%	1.7613%	2.2390%
	Time	2.33s + 14.16s	15.54s + 32.88s	23.64s + 68.74s	2.58s + 12.10s	3.89s + 32.36s	5.49s + 64.53s
$H = 3n$	Gap	-0.0001%	0.0032%	-0.0022%	0.5800%	1.7135%	2.2031%
	Time	2.33s + 14.14s	15.54s + 32.95s	23.64s + 68.84s	2.58s + 12.10s	3.89s + 32.64s	5.49s + 64.12s
$H = 5n$	Gap	-0.0001%	-0.0034%	-0.0021%	0.4192%	1.6388%	2.1371%
	Time	2.33s + 14.02s	15.54s + 32.85s	23.64s + 68.67s	2.58s + 12.30s	3.89s + 32.94s	5.49s + 65.95s
$H = 8n$	Gap	-0.0001%	-0.0034%	-0.0019%	0.4839%	1.5996%	2.1119%
	Time	2.33s + 14.05s	15.54s + 33.48s	23.64s + 69.63s	2.58s + 12.26s	3.89s + 33.06s	5.49s + 66.47s
$H = 10n$	Gap	-0.0001%	-0.0036%	-0.0024%	0.3043%	1.4050%	1.8280%
	Time	2.33s + 14.13s	15.54s + 34.29s	23.64s + 68.91s	2.58s + 12.56s	3.89s + 33.79s	5.49s + 66.29s
$H = 20n$	Gap	-0.0001%	0.0036%	-0.0015%	0.5347%	1.6288%	1.9297%
	Time	2.33s + 14.32s	15.54s + 34.70s	23.64s + 73.77s	2.58s + 12.69s	3.89s + 35.26s	5.49s + 69.42s
$H = 30n$	Gap	-0.0001%	-0.0033%	-0.0012%	0.5781%	1.6060%	2.0051%
	Time	2.33s + 14.37s	15.54s + 34.89s	23.64s + 75.92s	2.58s + 12.93s	3.89s + 39.72s	5.49s + 75.64s
$H = 50n$	Gap	-0.0001%	0.0048%	-0.0004%	0.5357%	1.4193%	1.9964%
	Time	2.33s + 14.54s	15.54s + 38.06s	23.64s + 81.36s	2.58s + 13.89s	3.89s + 39.68s	5.49s + 80.42s
$H = 100n$	Gap	-0.0001%	-0.0017%	0.0003%	0.6030%	1.5522%	2.0316%
	Time	2.33s + 14.98s	15.54s + 43.85s	23.64s + 95.30s	2.58s + 14.49s	3.89s + 50.14s	5.49s + 115.71s