BAYESIAN FLOW NETWORKS FOR COMBINATORIAL OPTIMIZATION: AN EDGE-BASED APPROACH TO THE TRAVELING SALESMAN PROBLEM

Anonymous authors
Paper under double-blind review

ABSTRACT

The Traveling Salesman Problem (TSP) is a fundamental combinatorial optimization problem, and non-autoregressive generative models have recently demonstrated strong solving capabilities on this problem. Diffusion-based solvers generate tours by iteratively denoising sampled edge configurations, but they do not retain the probabilities of alternative edge selections as the state is progressively refined. However, many locally plausible edge selections in the TSP continue to compete before their global compatibility becomes clear, creating a need for a generation state that can explicitly represent and progressively update such uncertainty. To address this, we introduce Bayesian Flow Networks (BFNs), which use probabilistic beliefs over alternative edge selections as the generative state and progressively refine these beliefs during generation, and propose Edge-BFN (EBFN), a non-autoregressive solver operating in the edge space. EBFN continuously maintains probabilistic beliefs throughout the iterative process, allowing competing edge selections to be progressively distinguished as global information accumulates, and uses the beliefs to generate and select multiple candidate solutions. Experiments on multiple TSP benchmarks under different test-time search budgets show that EBFN achieves competitive results and outperforms recent diffusion-based solvers in some settings, suggesting that directly refining probabilistic confidence over edge selections holds substantial potential for exploring competing decision alternatives.

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the classic problems in combinatorial optimization. Given a set of cities and the travel cost between every pair of cities, the TSP aims to find a minimum-cost Hamiltonian cycle that visits each city exactly once and returns to the starting city. As a representative combinatorial optimization problem with broad practical applications, the TSP has been studied in a wide range of real-world scenarios (Lenstra & Kan, 1975; Matai et al., 2010). Over the past decades, numerous exact algorithms (Applegate et al., 2011; Padberg & Rinaldi, 1991), approximation algorithms (Christofides, 2022; Arora, 1998; Karlin et al., 2021), and heuristic methods (Croes, 1958; Helsgaun, 2000; Johnson & McGeoch, 1997) have been developed for the TSP. However, these methods often rely on sophisticated search, pruning, or manually designed heuristic strategies, whose performance and applicability depend heavily on domain knowledge.

The rapid development of neural networks in recent years has stimulated research on Neural Combinatorial Optimization (NCO) (Wu et al., 2026; Bengio et al., 2021). Existing NCO methods can be broadly categorized into autoregressive (AR) and non-autoregressive (NAR) approaches (Sun & Yang, 2023). AR methods typically construct TSP tours sequentially in an autoregressive manner by selecting the next node at each step to form a complete solution, and have achieved strong solving performance (Vinyals et al., 2015; Bello et al., 2016; Kool et al., 2018; Kwon et al., 2020). However, such sequential generation requires multiple serial decisions, limiting inference efficiency (Joshi et al., 2019; Sun & Yang, 2023). Moreover, early decisions are committed and cannot be revised once made, which can make the generated tour sensitive to local mistakes (Barrett et al., 2020). In contrast, NAR methods generate multiple components of a solution in parallel, substantially reducing the inference overhead caused by sequential generation (Joshi et al., 2019; Sun &

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Yang, 2023). However, early NAR methods often struggle to achieve both efficient generation and high solution quality, and their solving performance remains behind that of AR methods in some settings (Xiao et al., 2025; Wang et al., 2025).

Diffusion-based TSP methods have achieved substantial improvements over conventional single-pass NAR approaches (Sun & Yang, 2023; Li et al., 2023; 2024; Zhao et al., 2025; Wang et al., 2025; Feng et al., 2026). These methods typically generate tours by iteratively denoising corrupted edge configurations. During this process, the model progressively refines a noisy realization of the edge variables rather than a probability distribution over edge choices. However, multiple locally plausible edge selections often need to be distinguished based on their global compatibility. This requires TSP solvers to continuously maintain and update the selection probabilities of different edges as global information is progressively accumulated.

To maintain edge-selection probabilities in the TSP, we propose Edge-BFN (EBFN). The model formulates adjacency matrix generation as an edge classification task and performs BFN generation in the edge space in a non-autoregressive manner. EBFN treats edge beliefs as the generation state, generates edge heatmaps through a GNN, and iteratively updates the beliefs while retaining the edge heatmaps from each step and decoding them into candidate solutions. To further explore the solution space, we apply Gumbel noise to the candidate heatmaps for stochastic sampling, generating additional candidate tours and obtaining the final solution.

We conduct extensive experiments on TSP instances with 20 to 500 nodes and compare EBFN with a variety of baseline methods. Experimental results show that EBFN achieves strong solution quality in most settings and reaches the best performance on some instances. On real-world TSP instances, EBFN demonstrates stronger performance, indicating its potential for generalization. Ablation studies further verify the effectiveness of the individual components of EBFN.

Our main contributions are summarized as follows:

- We develop a non-autoregressive BFN solver that explicitly maintains probabilistic beliefs over candidate edges and iteratively refines these beliefs over multiple flow steps while preserving parallel updates across edge variables.
- We retain the edge heatmaps generated at different inference steps as candidate solutions and use Gumbel perturbations to generate additional candidate tours, enabling further exploration of the solution space.
- We conduct experiments on TSP instances of different sizes and real-world instances. The results demonstrate that EBFN achieves strong solution quality and generalization performance.

2 Related Work

2.1 Autoregressive Neural Solvers

Autoregressive solvers construct a solution through a sequence of conditional decisions. Pointer Networks introduced attention-based generation of variable-length permutations and provided an early neural formulation for routing problems (Vinyals et al., 2015). Bello et al. cast the TSP as a reinforcement learning problem and trained a pointer policy with policy gradients (Bello et al., 2016). The Attention Model replaced recurrence with an attention encoder–decoder and became a standard backbone for neural routing solvers (Kool et al., 2018), and POMO exploited the multiple equivalent optima induced by routing symmetries to generate several solutions per instance (Kwon et al., 2020). Pointerformer further introduced a multi-pointer Transformer with feature augmentation (Jin et al., 2023). These methods can produce high-quality tours, especially when combined with sampling, beam search, or test-time search. However, each decision conditions on the previously constructed partial solution, so the inference cost grows with the number of construction steps and further with the number of candidates or search procedures. This sequential dependency motivates non-autoregressive solvers.

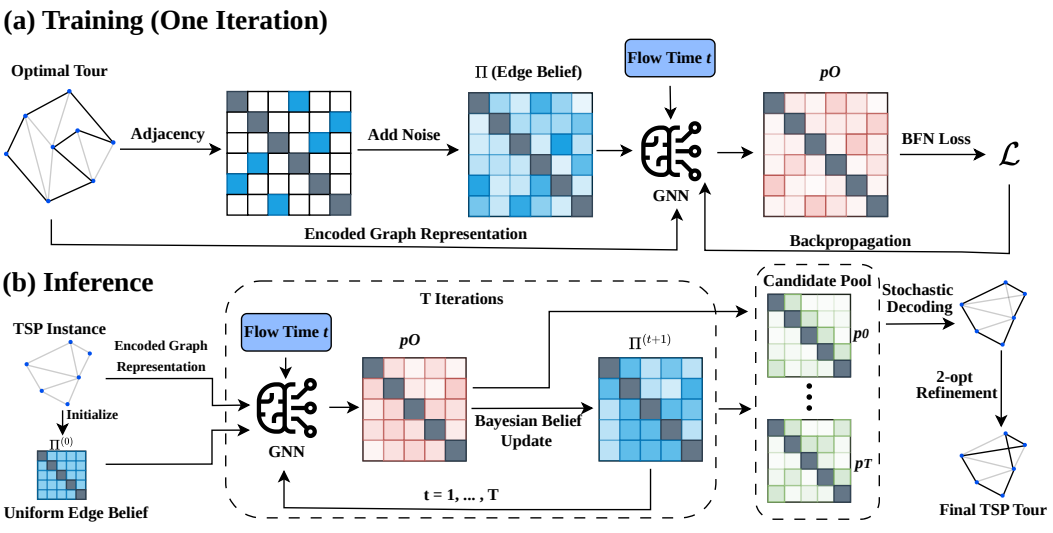

Figure 1: Overview of the proposed EBFN framework. (a) Training: optimal tours are converted into edge representations, perturbed to construct noisy beliefs, and used to train the network. (b) Inference: Edge beliefs are iteratively refined through Bayesian flow, with intermediate GNN-generated heatmaps retained for candidate generation. Gumbel perturbation and 2-opt are then used to explore and refine the generated tours.

2.2 Non-Autoregressive TSP Solvers

Non-autoregressive (NAR) solvers reduce this dependency by predicting solution-related variables in parallel. Joshi et al. trained a graph convolutional network to predict per-edge tour probabilities and decoded with highly parallel beam search, establishing the heatmap-based formulation (Joshi et al., 2019). Fu et al. combined heatmap prediction with Monte Carlo tree search to improve generalization to larger instances (Fu et al., 2021), and NAR4TSP trained a NAR solver with reinforcement learning while incorporating decoded solutions into training (Xiao et al., 2025). A complementary study examined how inductive bias, architecture, learning paradigm, and evaluation protocol affect the generalization of neural TSP models (Joshi et al., 2022). These works show that solution-related variables can be predicted in parallel, but obtaining a feasible Hamiltonian cycle may still require beam search, tree search, or another combinatorial decoder. Diffusion-based methods extend this line by replacing single-pass prediction with iterative refinement (Sun & Yang, 2023).

2.3 Diffusion-based Combinatorial Optimization

Diffusion models generate samples by reversing a stochastic corruption process, with DDPM (Ho et al., 2020) for the discrete-step view and score-based methods (Song et al., 2020) for the continuous-time view. For discrete variables, solvers use either discrete transitions, as in D3PM (Austin et al., 2021), or continuous relaxations. DIFUSCO applies iterative denoising to produce edge heatmaps for the TSP and the maximum independent set (Sun & Yang, 2023). T2TCO and Fast T2T combine learned solution distributions with test-time optimization (Li et al., 2023; 2024), DEITSP develops an efficient non-autoregressive solver with iterative inference (Wang et al., 2025), and unsupervised variants avoid optimal labels (Sanokowski et al., 2024). These methods iterate on a noisy realization of the edge variables and consume the network's per-step prediction in each reverse transition. Our method instead follows the Bayesian Flow Network formulation introduced next and carries the belief over edges as the state, updated by closed-form Bayesian inference, so that competing edge hypotheses remain explicit at every step.

2.4 BAYESIAN FLOW NETWORKS

Bayesian Flow Networks (BFNs) formulate generation as the iterative evolution of a probabilistic belief about the data rather than the denoising of corrupted samples (Graves et al., 2023). At each flow step a network predicts information about the underlying variables, and the belief is updated by a closed-form Bayesian rule. For discrete data, the belief is categorical, so neither a continuous relaxation nor a purpose-built discrete transition is required (Graves et al., 2023). The relationship between BFNs and diffusion models has been studied through stochastic differential equations, distinguishing the distribution parameters evolved by BFNs from the noisy samples evolved by diffusion models (Xue et al., 2024). BFN-based models have since been applied to structured generation such as three-dimensional molecules with geometric and categorical variables (Song et al., 2024). To our knowledge, BFNs have not been studied for the TSP or for the direct generation of edge variables in neural combinatorial optimization.

3 PRELIMINARIES

3.1 PROBLEM FORMULATION

Given a complete weighted graph $G = (V, E)$ with $N = |V|$ nodes and pairwise distances d_{ij}, the Traveling Salesman Problem (TSP) seeks a Hamiltonian cycle that visits every node exactly once and returns to the starting node while minimizing the total travel distance. We represent a candidate tour by a binary directed edge matrix

$$X \in \{0, 1\}^{N \times N}, \tag{1}$$

where $X_{ij} = 1$ indicates that the directed edge from node i to node j is selected. A feasible tour must satisfy the degree constraints

$$\sum_{j \neq i} X_{ij} = 1, \qquad \sum_{j \neq i} X_{ji} = 1, \tag{2}$$

for every node i, each node has exactly one outgoing and one incoming edge. Together with the requirement that all selected edges form a single connected cycle, these constraints ensure a valid Hamiltonian tour. The objective is

$$\min_{X} \sum_{i \neq j} X_{ij}\, d_{ij}. \tag{3}$$

In words, the TSP is the problem of choosing a sparse subset of edges that forms one connected cycle of minimum total length.

3.2 BAYESIAN FLOW NETWORKS

Bayesian Flow Networks (BFNs) generate data by progressively refining a probabilistic belief about the clean data, rather than by denoising a corrupted sample (Graves et al., 2023). For a discrete variable x taking one of C values, the belief is a categorical distribution $\boldsymbol{\pi} = [\pi_0, \ldots, \pi_{C-1}]$, where π_k is the current belief that $x = k$. Starting from an uninformative prior, each flow step maps the belief to a prediction p_O of the clean value, draws a noisy observation $\mathbf{y}$ of the true value at a prescribed accuracy α, and updates the belief by Bayes' rule. A larger α means a more informative observation and $\alpha = 0$ gives pure noise. Given $\mathbf{y}$, the update has the closed form

$$h(\boldsymbol{\pi}, \mathbf{y})_k = \frac{e^{y_k}\, \pi_k}{\sum_{k'=0}^{C-1} e^{y_{k'}}\, \pi_{k'}}, \tag{4}$$

and is applied independently to each factorized variable. The accuracy is governed by a monotone schedule $\beta(t)$ with $\beta(0) = 0$. In a discrete implementation with T steps, the accuracy at step i is

$$\alpha_i = \beta(t_i) - \beta(t_{i-1}), \qquad t_i = i/T. \tag{5}$$

We give the exact sender and receiver distributions, the Bayesian flow p_F, and the training objective in Appendix A.1.

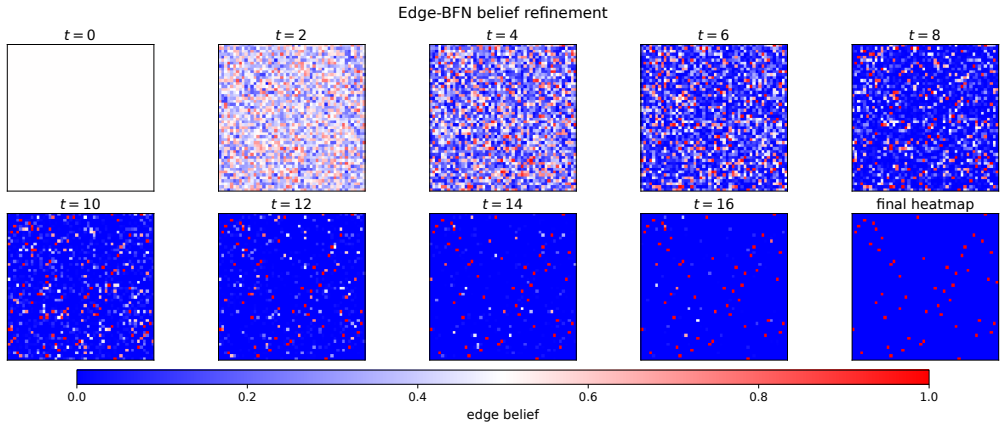

Figure 2: Illustration of the iterative belief update in EBFN. The edge beliefs are progressively refined through neural prediction and Bayesian updates.

4 METHOD

4.1 EDGE-BFN

Edge-BFN (EBFN) instantiates BFN on the TSP by treating every directed edge as a binary variable. The training and inference processes of EBFN are shown in Fig. 1. At a high level, EBFN starts from a uniform belief in which all edges are equally likely. It repeatedly lets a graph neural network predict how likely each edge is to belong to the optimal tour, and each prediction is folded back into the belief through a Bayesian update. After several steps the beliefs concentrate on a small set of edges. Because the beliefs remain probabilities rather than hard decisions, a single refinement pass yields a shared representation from which many candidate tours can be sampled.

Edge variables and beliefs. For every ordered pair of distinct nodes (i, j) we define an edge variable $x_{ij} \in \{0, 1\}$, where $x_{ij} = 1$ means that edge (i, j) belongs to the target tour. Because these variables are binary, $C = 2$, the categorical belief reduces to a single number. Writing $\pi^{(t)}_{ij,0} = 1 - \pi^{(t)}_{ij,1}$, the state of edge (i, j) is fully described by

$$\boldsymbol{\pi}^{(t)}_{ij} = \big[\pi^{(t)}_{ij,0},\ \pi^{(t)}_{ij,1}\big] = \big[1 - \pi^{(t)}_{ij,1},\ \pi^{(t)}_{ij,1}\big], \qquad \pi^{(t)}_{ij,1} \in [0, 1], \tag{6}$$

the belief that the edge belongs to the tour. We collect these scalars into the belief matrix and equivalently use the log-odds

$$\Pi^{(t)} = \big[\pi^{(t)}_{ij,1}\big]^{N}_{i,j=1}, \qquad \theta^{(t)}_{ij} = \operatorname{logit}\big(\pi^{(t)}_{ij,1}\big), \qquad \pi^{(t)}_{ij,1} = \operatorname{sigmoid}\big(\theta^{(t)}_{ij}\big). \tag{7}$$

We initialize $\theta^{(0)}_{ij} = 0$, i.e. $\pi^{(0)}_{ij,1} = 1/2$, with every edge equally likely at the start.

Network and prediction. The network is an anisotropic message-passing GNN f_ϕ over the TSP graph (Sun & Yang, 2023). Each node carries its normalized 2D coordinates, while each directed edge (i, j) carries the centered belief $2\pi^{(t)}_{ij,1} - 1$ and its length d_{ij}, with the flow time injected as an embedding. After L message-passing layers, a shared output head emits two logits per directed edge, so that

$$p^{(t)}_{ij} = p_O\big(k = 1 \mid \boldsymbol{\pi}^{(t)}_{ij}, t\big) = \big(\operatorname{softmax}(f_\phi(G, \Pi^{(t)}, t)_{ij})\big)_1, \tag{8}$$

where $p^{(t)}_{ij} \in [0, 1]$ is the predicted probability that edge (i, j) belongs to the tour. All edges are predicted in one forward pass. EBFN is therefore non-autoregressive. As we show below, this joint prediction is what allows the belief to exploit global structure rather than a fixed construction order.

4.2 TRAINING

Training follows the BFN sender/receiver scheme, with one binary variable per directed edge. Each optimal tour is encoded as its clean edge variables $x_{ij} \in \{0, 1\}$.

Accuracy schedule. The accuracy is governed by the schedule $\beta(t)$ of Eq. (5). We use the quadratic schedule

$$\beta(t) = \beta_1 t^2, \qquad \beta_1 = 20, \qquad \text{so} \qquad \alpha_s = \beta(t_s) - \beta(t_{s-1}) = \frac{\beta_1(2s-1)}{T^2}, \qquad t_s = s/T. \tag{9}$$

Both training and inference use this accuracy schedule α_s. At inference, the effective step size is additionally scaled by a step-size coefficient c (Section 4.3), whose effect we study in Appendix B.3.

Training beliefs and observations. Each training step involves two independent random draws at a sampled flow step s. First, the network input must be a belief at the appropriate noise level rather than the clean labels, so we draw it from the Bayesian flow distribution of Eq. (18). Specifically, with $\beta(t_s) = \beta_1 t_s^2$,

$$\boldsymbol{\pi}_{ij} = \mathrm{softmax}\Big(\beta(t_s)\left(C\mathbf{e}_{x_{ij}} - \mathbf{1}\right) + \sqrt{\beta(t_s)C}\,\boldsymbol{\xi}'_{ij}\Big), \qquad \boldsymbol{\xi}'_{ij} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{10}$$

which is equivalent to sampling $\mathbf{y}' \sim \mathcal{N}(\beta(t_s)(C\mathbf{e}_{x_{ij}} - \mathbf{1}), \beta(t_s)C\mathbf{I})$ and setting $\boldsymbol{\pi}_{ij} = \mathrm{softmax}(\mathbf{y}')$. The network then consumes $\boldsymbol{\pi}_{ij}$ through the centered belief $2\pi_{ij,1} - 1$. Second, to define the loss we draw an independent observation of the same edge state from the sender distribution at accuracy α_s,

$$\mathbf{y}_{ij} = \alpha_s\left(C\mathbf{e}_{x_{ij}} - \mathbf{1}\right) + \sqrt{\alpha_s C}\,\boldsymbol{\xi}_{ij}, \qquad \boldsymbol{\xi}_{ij} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{11}$$

with $C = 2$. The two draws use independent noise, where the first determines what the network sees and the second determines what the loss asks it to explain.

Loss. Given the belief $\boldsymbol{\pi}_{ij}$, the network outputs a categorical distribution $p_{O,ij} = [p_{O,ij,0}, p_{O,ij,1}]$. We minimize a single-step Monte-Carlo estimate of the sender–receiver divergence,

$$\begin{aligned} \mathcal{L}_{\mathrm{BFN}} &= \frac{1}{N^2}\sum_{i\neq j} \mathbb{E}_s\, \mathbb{E}_{\boldsymbol{\pi}_{ij}\sim p_F}\, \mathbb{E}_{\mathbf{y}_{ij}\sim p_S}\left[\ell_{ij}\right], \\ \ell_{ij} &= \log p_S(\mathbf{y}_{ij} \mid x_{ij}; \alpha_s) - \log p_R(\mathbf{y}_{ij} \mid \boldsymbol{\pi}_{ij}, t_s; \alpha_s)\,. \end{aligned} \tag{12}$$

For $C = 2$ the two Gaussian normalization constants cancel, so ℓ_{ij} takes the computable form

$$\begin{aligned} \ell_{ij} = &-\frac{\|\mathbf{y}_{ij} - \alpha_s(C\mathbf{e}_{x_{ij}} - \mathbf{1})\|^2}{2\alpha_s C} \\ &- \log\sum_{k=0}^{1} p_{O,ij,k}\, \exp\Big(-\frac{\|\mathbf{y}_{ij} - \alpha_s(C\mathbf{e}_k - \mathbf{1})\|^2}{2\alpha_s C}\Big). \end{aligned} \tag{13}$$

In words, the first term rewards the model when the true class explains the observation, and the second term penalizes probability assigned to the other class. The expression is a two-class cross-entropy with observation-dependent logits. In each training step we draw one s, $\boldsymbol{\pi}_{ij}$ and $\mathbf{y}_{ij}$ per example, and average ℓ_{ij} over all edges and the batch. The constant factor T of the discrete-time formulation is absorbed into the learning rate. Finally, with probability 0.25 we replace the input belief by a uniform distribution and set $t_s = 0$, so that the network also learns to predict from the coordinates alone. This ensures that the first inference step, which starts from a uniform prior, is well defined.

4.3 INFERENCE

At inference, EBFN starts from the uniform belief $\theta_{ij}^{(0)} = 0$ and refines the edge beliefs through the Bayesian flow process (Fig. 2). At each step the network predicts $p_{ij}^{(t)}$, after which the belief is updated in log-odds space by

$$\theta_{ij}^{(t+1)} = \theta_{ij}^{(t)} + c\,\alpha_t\left(2p_{ij}^{(t)} - 1\right), \qquad \alpha_t = \frac{\beta_1(2t-1)}{T^2}, \tag{14}$$

Table 1: Performance comparison on TSP20, TSP50, and TSP100. Gap (%) denotes the average relative gap to Concorde, and Time denotes total inference time. G, S, BS, 2OPT, and AUG denote Greedy search, Sampling, Beam search, 2-opt, and instance augmentation, respectively. † denotes evaluation on TSP20 and TSP50 using pre-trained TSP50 weights.

		TSP20		TSP50		TSP100	
Method	Type	Gap (%)	Time	Gap (%)	Time	Gap (%)	Time
DEITSP (16 Iter)	NAR, G, 2OPT	**0.00**	3.6m	<u>0.01</u>	3.8m	0.16	5.0m
DIFUSCO†	NAR, G, 2OPT	<u>0.15</u>	17.3m	0.10	20.0m	0.28	20.0m
T2TCO†	NAR, G, 2OPT	<u>0.15</u>	2.6m	0.09	2.9m	0.28	3.7m
Pointerformer	AR, G, AUG	**0.00**	13.4s	**0.00**	23.3s	0.17	1.0m
GCN	NAR, BS1280	0.17	3.5m	0.10	8.1m	2.13	16.4m
AM	AR, BS1280	**0.00**	5.0m	0.25	13.4m	2.48	2.4m
POMO	AR, G, AUG	**0.00**	18.3s	**0.00**	19.5s	<u>0.15</u>	10.6s
NAR4TSP†	NAR, BS1000	0.25	5.7s	0.26	4.1s	0.85	8.7s
EBFN (ours)	NAR, S, 2OPT	**0.00**	54.8s	**0.00**	1.8m	**0.11**	5.2m

where c is a step-size coefficient. In words, each edge's log-odds is moved towards the network's prediction by an amount proportional to the accuracy α_t and to c. Edges the network finds likely gain belief, while unlikely edges lose it.

Eq. (14) is a moment-matched (conditional-expectation) approximation to the BFN belief update (derivation in Appendix A.2). It replaces the random sender message in Eq. (4) by its conditional expectation under the receiver, which removes the sender noise. It is therefore an inference-time approximation rather than the exact BFN sampling procedure. The derivation produces a constant factor $C = 2$, which we absorb into c. Since $\sum_{t=1}^{T} \alpha_t = \beta_1$, the total change of any log-odds over a pass is bounded by $c\beta_1$ (Appendix A.2). The coefficient c thus controls how far the belief can move, and a smaller value yields finer belief refinement and better solutions under a limited sampling budget (Appendix B.3). Training does not use c. It keeps the accuracy schedule α_s so that the network learns to predict at every noise level. At inference the belief is already being sharpened, and we found that injected sender noise introduced additional fluctuations that lowered final solution quality. An ablation over the sender-noise scale is given in Appendix B.4.

The same update is applied to all directed edges in parallel. Each flow step requires one graph neural network evaluation followed by simultaneous belief updates. We retain the heatmap $P_t := [p_{ij}^{(t)}]$ produced by every network evaluation, obtaining a sequence $\{P_t\}_{t=0}^{T}$ of $T+1$ heatmaps. P_T is the final read-out of the belief. These intermediate heatmaps are retained because they provide a multi-stage representation of the solution space without additional network cost.

4.4 Multi-Stage Tour Decoding

The final belief matrix is an edge-wise confidence representation and does not by itself guarantee a feasible tour. We therefore convert the learned heatmaps into tours by stochastic decoding. Because the heatmap produced at every refinement step is retained, a single pass provides $T+1$ complementary read-outs at no additional network cost. For each heatmap we sample K candidates by adding Gumbel noise to its edge scores, map each candidate to a feasible tour with a subtour-avoiding greedy decoder (Sun & Yang, 2023), and refine it with 2-opt local search (Croes, 1958). The shortest tour over all heatmaps and candidates is reported. Appendix A.3 gives the exact scoring and decoding procedure.

5 Experiments

This section provides a comprehensive evaluation of EBFN. We first introduce the datasets, baseline methods, and model settings used in the experiments. We then compare EBFN with multiple baselines in terms of solution quality and running time. Next, we evaluate the generalization ability of EBFN on TSP instances with different sizes and distributions. Finally, we conduct ablation studies to analyze the contributions of different components of EBFN.

Table 2: Generalization performance of models trained on TSP100 and evaluated on TSP200 and TSP500. Gap (%) denotes the average relative performance gap to Concorde, and Time denotes the total running time over the entire test set. Bold and underlined values indicate the best and second-best solution quality, respectively.

		TSP200		TSP500	
Method	Type	Gap (%)	Time	Gap (%)	Time
DEITSP (16 Iter)	NAR, G, 2OPT	**0.48**	1.1m	**2.15**	7.0m
DIFUSCO	NAR, G, 2OPT	0.89	3.5m	3.85	25.9m
T2TCO	NAR, G, 2OPT	0.69	1.5m	3.22	1.1m
Pointerformer	AR, G, AUG	1.44	27.4s	12.56	1.4m
GCN	NAR, BS1280	48.94	3.6m	79.12	10.4m
AM	AR, BS1280	6.15	48.0s	18.03	4.7m
POMO	AR, G, AUG	1.64	3.9s	17.33	6.7s
EBFN (ours)	NAR, S, 2OPT	<u>0.56</u>	2.2m	<u>2.64</u>	18.8m

Table 3: Generalization performance on real-world TSP instances. Gap (%) denotes the average relative performance gap to Concorde, and Time denotes the total running time over the entire test set. Bold and underlined values indicate the best and second-best solution quality, respectively.

		USA		JA		BM		TSPLIB26	
Method	Type	Gap (%)	Time	Gap (%)	Time	Gap (%)	Time	Gap (%)	Time
DEITSP (16 Iter)	NAR, G, 2OPT	<u>1.14</u>	33.9s	<u>0.92</u>	36.5s	<u>2.46</u>	36.4s	<u>0.51</u>	11.3s
DIFUSCO	NAR, G, 2OPT	3.07	2.0m	2.53	2.1m	5.25	2.0m	1.02	16.6s
T2TCO	NAR, G, 2OPT	2.92	29.0s	2.53	29.0s	4.43	28.7s	0.84	28.7s
AM	AR, BS1280	79.52	16.9s	116.51	17.2s	208.83	17.3s	21.13	7.5s
POMO	AR, G, AUG	4.75	3.9s	7.80	3.9s	3.13	3.9s	1.50	68.1s
EBFN (ours)	NAR, S, 2OPT	**0.84**	42.3s	**0.31**	46.5s	**1.21**	45.3s	**0.33**	12.7s

5.1 Experimental Setup

Datasets. We use the TSP-20/50/100 training instances consistent with previous work, where all instances are uniformly generated in the range $[0, 1]$ (Kool et al., 2018; Kwon et al., 2020; Wang et al., 2025). The TSP-20/50/100/200/500 test sets contain the same instances as those used by DEITSP (Wang et al., 2025). To further evaluate the generalization ability of EBFN across different data distributions, following the setting of DEITSP, we randomly sample TSP instances with 100 nodes from the real-world datasets USA13509, BM33708, and JA9847, as well as from TSPLIB. The optimal solutions for all test instances are obtained using Concorde.

Baselines. We compare EBFN with several baseline methods, including DEITSP (Wang et al., 2025), DIFUSCO (Sun & Yang, 2023), T2TCO (Li et al., 2023), Pointerformer (Jin et al., 2023), AM (Kool et al., 2018), POMO (Kwon et al., 2020), NAR4TSP (Xiao et al., 2025), and GCN (Joshi et al., 2019). We use the average relative performance gap to Concorde (Gap) and total run time (Time) as the primary evaluation metrics. For the TSP-20/50/100 experiments, all methods are trained and evaluated at the same problem size. The models trained on TSP-100 are then directly evaluated on TSP-200/500 and the real-world datasets to assess their generalization ability. For methods without available model weights at a specific problem size, the weights trained at another problem size are used, as indicated in the corresponding tables.

Model Settings. Unless otherwise specified, all experiments use the same EBFN configuration. The GNN consists of 6 layers, with a hidden dimension of 384 in each layer. The number of refinement steps T is set to 20, the scaling coefficient of the inference step size α_t is set to 0.0125, the Gumbel temperature τ is set to 0.5, and the number of samples K is set to 8. Other baseline methods use their default settings. All experiments are conducted on an NVIDIA RTX 4090 D GPU.

5.2 Results

Table 1 compares EBFN with the baselines on the training data distribution. EBFN reaches a Gap of 0.00% on TSP-20 and TSP-50, matching the best baselines, and 0.11% on TSP-100, lower than

Table 4: Ablation study of the EBFN decoder under different refinement steps T on TSP100. Gap (%) and Time denote the optimality gap and total inference time, respectively.

Configuration	$T = 5$		$T = 10$		$T = 15$		$T = 20$		$T = 25$	
	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
Full EBFN	0.20	1.6m	0.15	2.8m	0.13	4.0m	0.11	5.2m	0.10	6.3m
w/o intermediate states	0.41	2.2m	0.32	5.0m	0.29	8.6m	0.28	13.3m	0.25	19.0m
w/o Gumbel sampling	0.66	1.6m	0.56	2.8m	0.53	4.0m	0.50	5.1m	0.49	6.4m

DEITSP (0.16%) and all other compared methods, showing stable quality as the problem size grows from TSP-20 to TSP-100. However, POMO reaches 0.15% in 10.6 seconds, only 0.04 percentage points worse and approximately $30\times$ faster, so the advantage of EBFN on TSP-100 lies mainly against the diffusion-based baselines.

Table 2 evaluates cross-scale generalization from TSP-100 to larger instances. EBFN obtains Gaps of 0.56% on TSP-200 and 2.64% on TSP-500, ranking second at both scales, with DEITSP lowest and ahead by 0.49 percentage points on TSP-500 while several other methods degrade more strongly. These results indicate that the learned edge representation and refinement process are not tied to the training scale. Inference takes 2.2 minutes on TSP-200 and 18.8 minutes on TSP-500, reflecting the additional computation of multi-stage refinement and candidate generation.

Table 3 evaluates generalization under a distribution shift from uniform synthetic instances to real-world datasets. EBFN attains the smallest Gap on all four datasets, 0.84% (USA), 0.31% (JA), 1.21% (BM), and 0.33% (TSPLIB26), with the largest improvement on BM. This shows that EBFN preserves competitive solution quality when the data distribution changes, and its low gaps together with moderate inference time make it effective on real-world TSP instances.

5.3 Ablation Study

We ablate the two inference mechanisms of EBFN on TSP-100 across different numbers of refinement steps. The w/o intermediate states variant uses only the final edge heatmap for candidate generation, followed by Gumbel sampling and 2-opt, while the w/o Gumbel sampling variant uses all intermediate heatmaps but decodes each deterministically with 2-opt and no Gumbel perturbation. Both variants match the sample budget of Full EBFN, and Table 4 reports the results. Full EBFN improves monotonically as the number of refinement steps increases and attains its lowest Gap at $T = 25$. It also outperforms both variants at every tested step, showing that multi-stage candidate generation from intermediate heatmaps and Gumbel-based stochastic exploration both contribute to solution quality.

6 Conclusion

We present EBFN, a non-autoregressive model based on Bayesian Flow Networks (BFNs) for solving the Traveling Salesman Problem (TSP). To model the edge structure of the TSP, we represent each candidate edge as a binary variable and use BFN to model and iteratively predict edge relationships. We further retain the edge heatmaps generated at different refinement steps and combine them with Gumbel noise to generate multiple candidate solutions, enhancing the exploration of potentially high-quality solutions. Experimental results show that EBFN achieves competitive solution quality across TSP instances of different sizes and on real-world instances, while also demonstrating strong generalization across problem scales and data distributions.

However, the current EBFN relies on multi-stage candidate generation and decoding to obtain high-quality solutions, which introduces considerable inference overhead. How to reduce unnecessary candidate computation while maintaining solution quality and exploration capability remains an important direction for future research. One possible direction is to gradually focus the exploration on promising candidate edges and regions of the solution space based on the edge probabilities and uncertainties produced at different refinement stages, thereby achieving more efficient search. Future work will also explore extending BFN to a broader range of optimization problems and investigating how to further improve its applicability and robustness.

AI USE STATEMENT

In this work, we used generative AI tools for drafting and editing portions of the manuscript and for assisting with literature search. We did not use generative AI tools to generate research ideas or experimental results. We also used generative AI tools to assist with code implementation and debugging. We have reviewed all AI-assisted work. LLM-assisted code was verified and tested by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the model architecture and training objective in Section 4 and provide the full BFN definitions and the decoding procedure in Appendices A.1 and A.3. All hyperparameters, datasets, and the exact inference configuration are reported in Section 5.1. The derivation of the moment-matched update is provided in Appendix A.2. Source code and the scripts used to generate all tables will be released as an anonymous downloadable supplement.

REFERENCES

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem: a computational study. In *The traveling salesman problem*. Princeton university press, 2011.

Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3243–3250, 2020.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Operations Research Forum*, volume 3, pp. 20. Springer, 2022.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6): 791–812, 1958.

Shengyu Feng, Tarun Suresh, and Yiming Yang. Unsupervised diffusion solver for combinatorial optimization via combinatorial adjoint matching. *arXiv preprint arXiv:2605.30920*, 2026.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Alex Graves, Rupesh Kumar Srivastava, Timothy Atkinson, and Faustino Gomez. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023.

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8132–8140, 2023.

David S Johnson and Lyle A McGeoch. The traveling salesman problem: a case study. *Local search in combinatorial optimization*, pp. 215–310, 1997.

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

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27:70–98, 2022. doi: 10.1007/s10601-022-09327-y.

Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 32–45, 2021.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in neural information processing systems*, 33:21188–21198, 2020.

Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:30179–30206, 2024.

Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25, 2010.

Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

Yuxuan Song, Jingjing Gong, Hao Zhou, Mingyue Zheng, Jingjing Liu, and Wei-Ying Ma. Unified generative modeling of 3d molecules with bayesian flow networks. In *International Conference on Learning Representations*, volume 2024, pp. 47851–47869, 2024.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Mingzhao Wang, You Zhou, Zhiguang Cao, Yubin Xiao, Xuan Wu, Wei Pang, Yuan Jiang, Hui Yang, Peng Zhao, and Yuanshu Li. An efficient diffusion-based non-autoregressive solver for traveling salesman problem. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1469–1480, 2025.

Xuan Wu, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L Maskell, You Zhou, and Di Wang. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

Yubin Xiao, Di Wang, Boyang Li, Huanhuan Chen, Wei Pang, Xuan Wu, Hao Li, Dong Xu, Yanchun Liang, and You Zhou. Reinforcement learning-based nonautoregressive solver for traveling salesman problems. *IEEE Transactions on Neural Networks and Learning Systems*, 36(7):13402–13416, 2025.

Kaiwen Xue, Yuhao Zhou, Shen Nie, Xu Min, Xiaolu Zhang, Jun Zhou, and Chongxuan Li. Unifying bayesian flow networks and diffusion models through stochastic differential equations. *arXiv preprint arXiv:2404.15766*, 2024.

Hang Zhao, Kexiong Yu, Yuhang Huang, Renjiao Yi, Chenyang Zhu, and Kai Xu. Disco: Efficient diffusion solver for large-scale combinatorial optimization problems. *Graphical Models*, 141: 101284, 2025.

A Appendix

A.1 Bayesian Flow Networks

Output distribution. At flow step t, a neural network maps the current belief and any side information to the parameters of an output distribution

$$p_O(k \mid \boldsymbol{\pi}, t) = \big(\mathrm{softmax}(f_\phi(\boldsymbol{\pi}, t))\big)_k, \tag{15}$$

which encodes the model's guess for the clean value.

Sender and receiver. At accuracy $\alpha > 0$, the sender produces a noisy observation of the true value,

$$p_S(\mathbf{y} \mid x; \alpha) = \mathcal{N}(\mathbf{y} \mid \alpha\,(C\mathbf{e}_x - \mathbf{1})\,,\ \alpha C\mathbf{I})\,, \tag{16}$$

where $\mathbf{e}_x$ is the one-hot encoding of x. A larger α means a less noisy and more informative observation. The receiver expresses what the observation would look like for each possible value under the model's prediction, by convolving p_O with the same noise,

$$p_R(\mathbf{y} \mid \boldsymbol{\pi}, t; \alpha) = \sum_{k=0}^{C-1} p_O(k \mid \boldsymbol{\pi}, t)\, \mathcal{N}(\mathbf{y} \mid \alpha\,(C\mathbf{e}_k - \mathbf{1})\,,\ \alpha C\mathbf{I})\,. \tag{17}$$

Bayesian flow. The belief at time t follows the Bayesian flow distribution $p_F(\cdot \mid x; t)$, obtained by applying the update of Eq. (4) to the prior with total accuracy $\beta(t)$. For discrete data it can be sampled in closed form,

$$p_F(\boldsymbol{\pi} \mid x; t) = \mathbb{E}_{\mathbf{y} \sim \mathcal{N}(\beta(t)(C\mathbf{e}_x - \mathbf{1}),\ \beta(t)C\mathbf{I})} \big[\delta\,(\boldsymbol{\pi} - \mathrm{softmax}(\mathbf{y}))\big], \tag{18}$$

that is, by drawing $\mathbf{y} \sim \mathcal{N}(\beta(t)(C\mathbf{e}_x - \mathbf{1}), \beta(t)C\mathbf{I})$ and setting $\boldsymbol{\pi} = \mathrm{softmax}(\mathbf{y})$. As $t \to 1$, the belief concentrates on the clean value x.

Training objective. The network is trained so that the receiver matches the sender. For a clean value x and accuracy α, the loss is the Kullback–Leibler divergence from the sender to the receiver,

$$\mathcal{L}(x) = \mathbb{E}_t\, \mathbb{E}_{\boldsymbol{\pi} \sim p_F(\cdot \mid x; t)}\, \mathbb{E}_{\mathbf{y} \sim p_S(\cdot \mid x; \alpha_t)} \Big[\log p_S(\mathbf{y} \mid x; \alpha_t) - \log p_R(\mathbf{y} \mid \boldsymbol{\pi}, t; \alpha_t)\Big], \tag{19}$$

where the expectation over $\boldsymbol{\pi}$ reflects that the network input is itself a belief derived from the clean data. The first term does not depend on the network parameters, and minimizing $\mathcal{L}$ pushes p_R towards the observation distribution of the true value, so that p_O assigns high probability to the correct value. A useful property of BFN is that the network operates on distribution parameters on the probability simplex that are natively differentiable, rather than directly on noisy data samples.

A.2 Derivation of the Moment-Matched Belief Update

Setup. We consider a single binary edge variable $x \in \{0, 1\}$ and drop the edge indices (i, j) for clarity. The derivation applies edge-wise. At flow step t the belief is a categorical distribution

$$\boldsymbol{\pi}^{(t)} = [\pi_0^{(t)}, \pi_1^{(t)}], \qquad \pi_0^{(t)} + \pi_1^{(t)} = 1, \tag{20}$$

and we write $\lambda^{(t)} = \log(\pi_1^{(t)}/\pi_0^{(t)})$ for its log-odds, so that $\pi_1^{(t)} = \sigma(\lambda^{(t)})$ with σ the sigmoid.

Following BFN, the sender produces a noisy observation of the true value at accuracy $\alpha > 0$,

$$p_S(\mathbf{y} \mid x; \alpha) = \mathcal{N}(\mathbf{y} \mid \alpha(C\mathbf{e}_x - \mathbf{1}),\, \alpha C\mathbf{I}), \qquad \mathbf{y} \in \mathbb{R}^C, \tag{21}$$

the network predicts the clean class with probabilities $p_O = \mathrm{softmax}(f_\phi(\boldsymbol{\pi}, t))$, and the receiver is the convolution of the output distribution with the same observation noise,

$$p_R(\mathbf{y} \mid \boldsymbol{\pi}, t; \alpha) = \sum_{k=0}^{C-1} p_{O,k}\, \mathcal{N}(\mathbf{y} \mid \alpha(C\mathbf{e}_k - \mathbf{1}),\, \alpha C\mathbf{I}). \tag{22}$$

The exact BFN belief update is

$$h(\boldsymbol{\pi}, \mathbf{y})_k = \frac{\pi_k\, p_S(\mathbf{y} \mid x = k; \alpha)}{\sum_{k'} \pi_{k'}\, p_S(\mathbf{y} \mid x = k'; \alpha)}. \tag{23}$$

Exact update reduces to a softmax of the message. Because the sender is Gaussian,

$$\log p_S(\mathbf{y} \mid x = k; \alpha) = -\frac{\|\mathbf{y} - \alpha(C\mathbf{e}_k - \mathbf{1})\|^2}{2\alpha C} + \text{const.} \tag{24}$$

Expanding the squared norm,

$$\|\mathbf{y} - \alpha(C\mathbf{e}_k - \mathbf{1})\|^2 = \|\mathbf{y}\|^2 - 2\alpha\Big(Cy_k - \sum_j y_j\Big) + \alpha^2 C(C-1), \tag{25}$$

where the last term does not depend on k. Dividing Eq. (25) by $-2\alpha C$ and dropping all k-independent terms gives

$$\log p_S(\mathbf{y} \mid x = k; \alpha) = y_k + \text{const}, \tag{26}$$

so the exact update in Eq. (23) simplifies to

$$\pi'_k \propto \pi_k\, e^{y_k}. \tag{27}$$

Conditional expectation of the message. The exact update requires sampling $\mathbf{y}$. We instead replace $\mathbf{y}$ by its conditional expectation under the receiver,

$$\bar{\mathbf{y}} = \mathbb{E}_{p_R}[\mathbf{y}] = \sum_{k=0}^{C-1} p_{O,k}\, \alpha(C\mathbf{e}_k - \mathbf{1}) = \alpha\,(Cp_O - \mathbf{1}), \tag{28}$$

using $\sum_k p_{O,k} = 1$, where $p_O = (p_{O,0}, \ldots, p_{O,C-1})$.

Moment-matched update. Substituting Eq. (28) into Eq. (27) gives

$$\pi'_k \propto \pi_k\, e^{\alpha(Cp_{O,k} - 1)} \propto \pi_k\, e^{\alpha C p_{O,k}}, \tag{29}$$

since $e^{-\alpha}$ is independent of k and cancels in the normalization. Taking logarithms,

$$\log \pi'_k = \log \pi_k + \alpha C\, p_{O,k} + \text{const}. \tag{30}$$

For a binary variable $C = 2$, subtracting the two components gives the log-odds update

$$\lambda' = \lambda + \alpha C\,(p_{O,1} - p_{O,0}) = \lambda + C\alpha\,(2p_{O,1} - 1), \tag{31}$$

using $p_{O,0} = 1 - p_{O,1}$. Three properties follow. The log-odds increases when the network favours the edge ($p_{O,1} > \frac{1}{2}$) and decreases otherwise. The magnitude of the step is proportional to the network's confidence $|2p_{O,1} - 1|$. An uninformative prediction ($p_{O,1} = \frac{1}{2}$) leaves the belief unchanged.

Table 5: Effect of the Gumbel temperature τ (TSP-100).

τ	0	0.1	0.3	0.5	0.7	1.0
Gap (%)	0.501	0.278	0.152	0.108	0.093	**0.090**

Table 6: Effect of the number of candidates K (TSP-100).

K	1	2	4	8	16
Candidates	21	42	84	168	336
Gap (%)	0.313	0.216	0.146	0.108	**0.081**

Step-size coefficient. The constant factor $C = 2$ produced by the derivation is absorbed into a step-size coefficient c, and the inference update is

$$\lambda^{(t+1)} = \lambda^{(t)} + c\,\alpha_t \left(2p_{O,1}^{(t)} - 1\right), \qquad \alpha_t = \frac{\beta_1(2t-1)}{T^2}, \tag{32}$$

where α_t is the accuracy schedule and c is a tunable coefficient. Since $\sum_{t=1}^{T}(2t-1) = T^2$, the schedule satisfies $\sum_{t=1}^{T}\alpha_t = \beta_1$. We treat c as a hyperparameter and study its effect in Appendix B.3.

Relation to standard BFN sampling. Standard BFN inference draws $\mathbf{y} \sim p_R$ at every step and produces a stochastic trajectory. Our inference instead substitutes the conditional expectation in Eq. (28), which removes the sender noise and makes the trajectory deterministic. This substitution is an inference-time approximation to the BFN Bayesian update. It is not equivalent to the exact BFN sampling procedure. The network is nonetheless trained with sampled messages (Eqs. (10) and (11)), so it is exposed to this noise during training. Empirically, removing the sender noise at inference lowers the gap at every number of steps, and the corresponding ablation is reported in Appendix B.4.

A.3 Multi-Stage Tour Decoding

A single EBFN inference pass retains the heatmap $P_t = [p_{ij}^{(t)}]$ produced at each refinement step and therefore yields $T+1$ heatmaps. We decode each heatmap as follows.

Edge scores and Gumbel perturbations. We symmetrize the heatmap and convert it to log-odds,

$$\bar{P}_t = \tfrac{1}{2}\left(P_t + P_t^{\top}\right), \qquad z_t = \operatorname{logit}\big(\operatorname{clip}\big(\bar{P}_t, 10^{-6}, 1-10^{-3}\big)\big). \tag{33}$$

Symmetrization reflects that a tour uses undirected edges, and the clipping avoids infinite logits. We then draw K stochastic candidates per heatmap by adding Gumbel noise,

$$S_{t,k} = z_t + \tau\, G_{t,k}, \qquad G_{t,k} \sim \mathrm{Gumbel}(0,1), \qquad k = 1, \ldots, K, \tag{34}$$

where τ controls the noise level of the perturbation.

Constrained decoding. Each perturbed score matrix $S_{t,k}$ is mapped to a feasible tour with a greedy subtour-avoiding decoder (Sun & Yang, 2023). Candidate edges are considered in descending order of score, and an edge is inserted only if its two endpoints are currently in different paths, which prevents premature subtours. The procedure terminates when all nodes form a single Hamiltonian cycle. Each decoded tour is then improved by batched 2-opt local search (Croes, 1958), run to convergence on the GPU for every candidate.

Candidate pooling and selection. Writing $\mathcal{C}_{\mathrm{inter}} = \{(t,k)\}_{t=0}^{T}{}_{k=1}^{K}$ for all Gumbel candidates, the reported EBFN solution is

$$\hat{X} = \arg\min_{X \in \mathcal{C}_{\mathrm{inter}}} \mathrm{len}(X), \tag{35}$$

after 2-opt refinement.

Table 7: Effect of the step-size coefficient c (TSP-100).

c	0.0125	0.125	0.25	0.5	0.625	0.75	0.875	1.0
Gap (%)	**0.108**	0.136	0.145	0.157	0.164	0.168	0.169	0.177

Table 8: Moment-matched update versus the original BFN update (gap %).

	TSP-20	TSP-50	TSP-100
Moment-matched update	0.000	0.000	**0.108**
Original BFN update	0.000	0.000	0.222

B Additional Ablation and Sensitivity Analysis

We ablate the inference components of EBFN and study the sensitivity of its main hyperparameters on TSP-100, changing one quantity at a time. Unless stated otherwise, we use the configuration $T = 20$, $K = 8$, $\tau = 0.5$, step-size coefficient $c = 0.0125$ and all intermediate states.

B.1 Gumbel Temperature

We vary the Gumbel temperature τ with $T = 20$, $K = 8$, and all intermediate states. Setting $\tau = 0$ disables the perturbation and decodes each heatmap deterministically.

Gumbel exploration is a core component. Without it ($\tau = 0$) the gap degrades to 0.501%, whereas any $\tau > 0$ yields 0.09–0.11%. Within this setting the gap improves monotonically with τ and is lowest at $\tau = 1.0$ (0.090%).

B.2 Number of Candidates

We vary K with $T = 20$ and $\tau = 0.5$. The number of candidates is $|\mathcal{S}| \cdot K = 21K$, since each of the 21 states contributes K Gumbel samples.

The gap improves monotonically with diminishing returns as K grows, decreasing from 0.313% at $K = 1$ to 0.081% at $K = 16$. No saturation is observed within the tested range, and $K = 16$ remains better than $K = 8$, so the candidate budget can be increased further when more computation is available. We use $K = 8$ in the low-budget setting.

B.3 Step-Size Coefficient

We vary the step-size coefficient c of the moment-matched update with $T = 20$, $K = 8$, and $\tau = 0.5$. The gap decreases monotonically as c decreases, from 0.177% at $c = 1.0$ to 0.108% at $c = 0.0125$. A smaller coefficient makes each belief update gentler, so the read-outs change more slowly across refinement steps and the trajectory behaves like a finer multi-stage ensemble, which helps under a limited candidate budget.

B.4 Sender Noise

We compare the moment-matched update used in EBFN with the original stochastic BFN update that injects sender noise, at three problem sizes, using the corresponding trained model for each size. At TSP-20 and TSP-50 both updates reach a gap of 0.000%. At TSP-100 the moment-matched update is clearly better (0.108% versus 0.222%). With the small step-size coefficient the injected noise dominates the update, which acts as additional exploration at small sizes but degrades candidate quality at larger sizes. We therefore use the moment-matched update.