Neural Network Solvers for Combinatorial Optimization

Graph 10. Non-autoregressive CO Solvers

1

Lectures on Neural-network CO Solvers

- Graph 9: Autoregressive (AR) CO Solvers
- Graph 10: Non-autoregressive (NAR) CO Solvers
  - Reinforcement learning with DIMES [R Qiu*, Z Sun* & Y Yang, **NearIPS 2022**]
  - Supervised learning with DIFUSCO [Z Sun & Y Yang, **NearIPS 2023**]
- Graph 11: Pre-trained Large Language Models for CO
- Graph 12: Neural Solvers for Mixed Integer Programming

2

DIMES: A differentiable meta solver for CO

[Qiu*, Sun* & Yang, NearIPS 2022]

Key Ideas

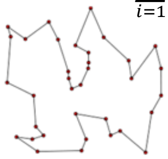
1) A unified framework for CO problems (including TSP and MIS)
2) A NAR CO solver with an encoder-only architecture (removing the expensive decoding parts in the AR solvers), scaled to graphs with $n = 10{,}000$ nodes
3) Using meta learning (in the training-phase) to enhance the accuracy on new graphs

Unified Framework for CO Problems

- Convert any CO problem into a task of {0,1}-vector generation
  - Let $\mathcal{F}_s = \{0,1\}^N$ be the space of feasible solutions for instance graph s;
  - Let $c_s\colon \mathcal{F}_s \to \mathbb{R}$ be the cost function for any solution $\boldsymbol{f} = (f_1, f_2, \ldots, f_N) \in \mathcal{F}_s$.

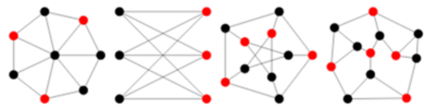
TSP: Finding the optimal tour with $\boldsymbol{f} \in \{0,1\}^N$ by selecting n edges from the total of $N = n^2$ edges

$$c_s^{\mathrm{TSP}}(\boldsymbol{f}) = \sum_{i=1}^{N} f_i \cdot length(f_i, s)$$



MIS: Finding the maximal subset with $\boldsymbol{f} \in \{0,1\}^N$ by selecting a subset from the total of $N = n$ nodes.

$$c_s^{\mathrm{MIS}}(\boldsymbol{f}) = \sum_{i=1}^{N} (1 - f_i)$$



Objective

- We want to minimize the expected cost of system-generated solutions for graph s

$$\min_{\theta} \mathcal{L}_\theta(s) = \min_{\theta} \sum\nolimits_{\pi \sim p_\theta(.|s)} [c_s(\pi)\ \ p_\theta(\pi|s)] \qquad \text{(previous lecture)}$$

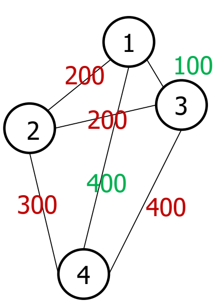
- Optimizing θ via reinforcement learning (RL) with

$$\nabla \mathcal{L}_\theta(s) = \sum\nolimits_{\pi \sim p_\theta(.|s)} c_s(\pi) \nabla \log\ p_\theta(\pi|s) \qquad \text{(previous lecture)}$$

- In AR model, $p_\theta(.|s)$ is estimated by the decoder. But now we want to eliminate the decode!
- Idea:
  - Training an encoder-only neural network to produce a heatmap (of node/edge weights based on training examples of <tour, cost> pairs.
  - Using the heatmap to conduct a search of plausible solutions during testing.

A toy example

Graph s



Feasible Solutions & Costs $(n! = 4 \cdot 3 \cdot 2 \cdot 1 = 24)$

$\pi_1 = 1 \to 2 \to 3 \to 4 \to 1, \quad c(\pi_1) = 200 + 200 + 400 + 400 = 1200$

$\pi_2 = 1 \to 2 \to 4 \to 3 \to 1, \quad c(\pi_2) = 200 + 300 + 400 + 100 = 1000$

$\pi_3 = \cdots$

$$\{(\pi_i, c_i\,)\} \xrightarrow{intuitively} Is\ e_{24} \text{ beter than } \mathrm{e}_{23}?$$

$$\{(\pi_i, c_i\,)\} \xrightarrow{RL\ training} Heatmap = \{edge\ weights\}$$

Toy Example: NAR Sequence Generation in Testing Phase

Heatmap Θ

	1	2	3	4
1		9	10	1
2	10		5	6
3	7	1		8
4	8	1	1	

$P_\theta(e_{ij} \in \pi^*|s) \propto \Theta_{ij}$

Sampled Tours based on Θ

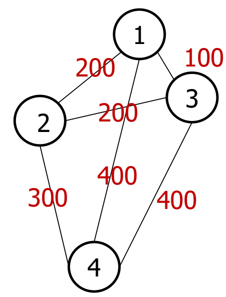
π_1: $e_{13} \rightarrow e_{34} \rightarrow e_{42} \rightarrow e_{21}$

$c(\pi_1) = 100 + 400 + 300 + 200 = 1000$

π_2: $e_{21} \rightarrow e_{13} \rightarrow e_{34} \rightarrow e_{42}$

$c(\pi_2) = 200 + 100 + 400 + 300 = 1000$

$\vdots$

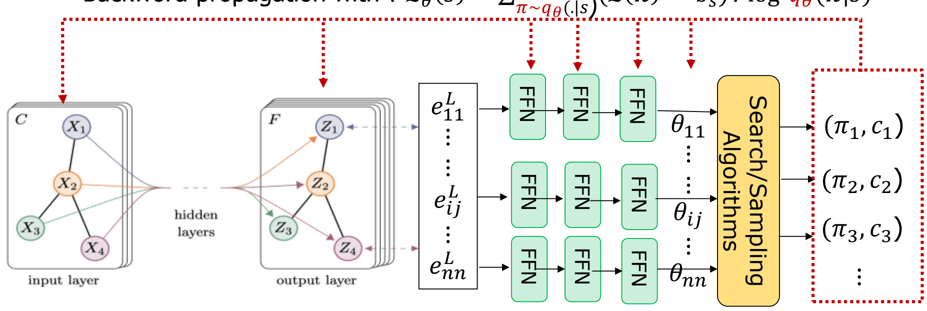


7

How do we obtain the heatmap?

GNN+MLP with RL over a training set of graphs and sampled tours per graph

Backword propagation with $\nabla \mathcal{L}_\theta(s) = \sum_{\pi \sim q_\theta(.|s)} (\mathcal{L}(\boldsymbol{\pi}) - b_s) \nabla \log q_\theta(\boldsymbol{\pi}|s)$



Anisotropic GNN
(for node & edge embeddings)

MLP
(mapping each edge vector to a scalar)

8

Anisotropic Graph Neural Networks

producing edge embedding for solving TSP (Z Sun & Y Yang, NeurIPS 2022)

Anisotropic Graph Neural Networks We follow Joshi et al. [33] on the choice of neural architectures. The backbone of the graph neural network is an anisotropic GNN with an edge gating mechanism [9]. Let $\boldsymbol{h}_i^\ell$ and $\boldsymbol{e}_{ij}^\ell$ denote the node and edge features at layer ℓ associated with node i and edge ij, respectively. The features at the next layer is propagated with an anisotropic message passing scheme:

$$\boldsymbol{h}_i^{\ell+1} = \boldsymbol{h}_i^\ell + \alpha(\mathrm{BN}(\boldsymbol{U}^\ell \boldsymbol{h}_i^\ell + \mathcal{A}_{j\in\mathcal{N}_i}(\sigma(\boldsymbol{e}_{ij}^\ell) \odot \boldsymbol{V}^\ell \boldsymbol{h}_j^\ell))), \tag{18}$$

$$\boldsymbol{e}_{ij}^{\ell+1} = \boldsymbol{e}_{ij}^\ell + \alpha(\mathrm{BN}(\boldsymbol{P}^\ell \boldsymbol{e}_{ij}^\ell + \boldsymbol{Q}^\ell \boldsymbol{h}_i^\ell + \boldsymbol{R}^\ell \boldsymbol{h}_j^\ell)). \tag{19}$$

where $\boldsymbol{U}^\ell, \boldsymbol{V}^\ell, \boldsymbol{P}^\ell, \boldsymbol{Q}^\ell, \boldsymbol{R}^\ell \in \mathbb{R}^{d\times d}$ are the learnable parameters of layer ℓ, α denotes the activation function (we use SiLU [15] in this paper), BN denotes the Batch Normalization operator [30], $\mathcal{A}$ denotes the aggregation function (we use mean pooling in this paper), σ is the sigmoid function, $\odot$ is the Hadamard product, and $\mathcal{N}_i$ denotes the outlinks (neighborhood) of node i. We use a 12-layer GNN with width 32.

Model Training

- **Loss Function**

$$\mathcal{L}_\Phi(\mathcal{C}) = \sum_{s\in\mathcal{C}} l_q(\theta_s) = \sum_{s\in\mathcal{C}} l_q\left[F_\phi(X_{s,}, A_s)\right]$$

where $\mathcal{C}$ is the training set of graphs and $s \in \mathcal{C}$ is an instance graph;

Φ is the neural network (GNN+MLP) parameter set;

$\theta_s \coloneqq F_\phi(X_{s,}, A_s)$ is the heatmap produced by the neural network given input graph s;

X_s is the matrix of node features and A_s is the adjacency matrix of graph s;

$q = q(\theta_s)$ is the auxiliary distribution for efficient sampling of solutions from heatmap θ_s .

- **Gradient**

$$\nabla_\Phi \mathcal{L}_\Phi(\mathcal{C}) = E_{s\in\mathcal{C}}\left[\nabla_\Phi \theta_s \cdot \nabla_{\theta_s} l_q(\theta_s)\right] = E_{s\in\mathcal{C}}\left[\nabla_\Phi F_\phi(X_{s,}, A_s) \cdot \nabla_{\theta_s} l_q(\theta_s)\right]$$

- **Forward/backward propagations**: updating Φ with the full formula and θ with the green part.

Results

Method	Type	TSP-500 Length ↓	TSP-500 Drop ↓	TSP-500 Time ↓	TSP-1000 Length ↓	TSP-1000 Drop ↓	TSP-1000 Time ↓	TSP-10000 Length ↓	TSP-10000 Drop ↓	TSP-10000 Time ↓
Concorde	OR (exact)	16.55*	—	37.66m	23.12*	—	6.65h	N/A	N/A	N/A
Gurobi	OR (exact)	16.55	0.00%	45.63h	N/A	N/A	N/A	N/A	N/A	N/A
LKH-3 (default)	OR	16.55	0.00%	46.28m	23.12	0.00%	2.57h	71.77*	—	8.8h
LKH-3 (less trails)	OR	16.55	0.00%	3.03m	23.12	0.00%	7.73m	71.79	—	51.27m
Nearest Insertion	OR	20.62	24.59%	0s	28.96	25.26%	0s	90.51	26.11%	6s
Random Insertion	OR	18.57	12.21%	0s	26.12	12.98%	0s	81.85	14.04%	4s
Farthest Insertion	OR	18.30	10.57%	0s	25.72	11.25%	0s	80.59	12.29%	6s
EAN	RL+S	28.63	73.03%	20.18m	50.30	117.59%	37.07m	N/A	N/A	N/A
EAN	RL+S+2-OPT	23.75	43.57%	57.76m	47.73	106.46%	5.39h	N/A	N/A	N/A
AM	RL+S	22.64	36.84%	15.64m	42.80	85.15%	63.97m	431.58	501.27%	12.63m
AM	RL+G	20.02	20.99%	1.51m	31.15	34.75%	3.18m	141.68	97.39%	5.99m
AM	RL+BS	19.53	18.03%	21.99m	29.90	29.23%	1.64h	129.40	80.28%	1.81h
GCN	SL+G	29.72	79.61%	6.67m	48.62	110.29%	28.52m	N/A	N/A	N/A
GCN	SL+BS	30.37	83.55%	38.02m	51.26	121.73%	51.67m	N/A	N/A	N/A
POMO+EAS-Emb	RL+AS	19.24	16.25%	12.80h	N/A	N/A	N/A	N/A	N/A	N/A
POMO+EAS-Lay	RL+AS	19.35	16.92%	16.19h	N/A	N/A	N/A	N/A	N/A	N/A
POMO+EAS-Tab	RL+AS	24.54	48.22%	11.61h	49.56	114.36%	63.45h	N/A	N/A	N/A
Att-GCN	SL+MCTS	16.97	2.54%	2.20m	23.86	3.22%	4.10m	74.93	4.39%	21.49m
DIMES (ours)	RL+G	18.93	14.38%	0.97m	26.58	14.97%	2.08m	86.44	20.44%	4.65m
	RL+AS+G	17.81	7.61%	2.10h	24.91	7.74%	4.49h	80.45	12.09%	3.07h
	RL+S	18.84	13.84%	1.06m	26.36	14.01%	2.38m	85.75	19.48%	4.80m
	RL+AS+S	17.80	7.55%	2.11h	24.89	7.70%	4.53h	80.42	12.05%	3.12h
	RL+MCTS	16.87	1.93%	2.92m	23.73	2.64%	6.87m	74.63	3.98%	29.83m
	RL+AS+MCTS	**16.84**	**1.76%**	2.15h	**23.69**	**2.46%**	4.62h	**74.06**	**3.19%**	3.57h

OR: Traditional Solver

RL: Reinforce. Learners

SL: Supervised Learners

Sampling Strategies:

G: Greedy Readout

S: Sampling

BS: Beam Search

AS: Active Search

MCTS: Monte-Carlo Tree Search

Outline of 3 Lectures: Graph 9, 10 and 11

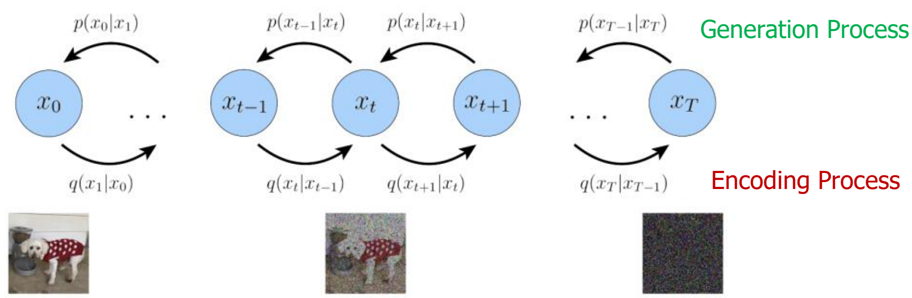
- Introduction to ML for Combinatorial Optimization (CO)
- Autoregressive (AR) CO Solvers
- Non-autoregressive (NAR) CO Solvers
  - Reinforcement learning with DIMES [R Qiu*, Z Sun* & Y Yang, **NearIPS 2022**]
  - Supervised learning with DIFUSCO [Z Sun & Y Yang, **NearIPS 2023**]
- Pre-trained Large Language Models [C Yang et al., ICLR 2024]

DIFUSCO: Graph-based Diffusion Solvers for CO
[Z Sun & Y Yang, NeurIPS 2023]

- Borrowing the key idea from recent diffusion models for computer vision
- Developed the 1st diffusion-based solver for CO problems
- Beat other SOTA CO solvers (e.g., DIMES) in both accuracy and efficiency

17

Diffusion modeling for generating high-quality images

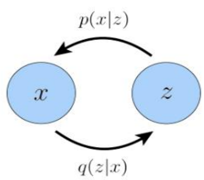


Rationale: Decomposing a hard problem ($x_T \rightarrow x_0$) into many easier problems ($x_t \rightarrow x_{t-1}$)

18

Recap: Variational Autoencoder (VAE)

generation



encoding

Evidence Lower Bound (ELBO)

$$\log p(x) = log \int p(x,z)dz \qquad z \sim \mathcal{N}(.)$$

$$= log \int \frac{p(x,z)q(z|x)}{q(z|x)} dz \qquad \text{(Multiplly by } \frac{q(Z|X)}{q(Z|X)} = 1)$$

$$= log \mathbb{E}_{q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] \qquad \text{(Definition of Expectation)}$$

$$\geq \mathbb{E}_{q(z|x)}\left[log \frac{p(x,z)}{q(z|x)}\right] \qquad \text{(Jensen's Inequality)}$$

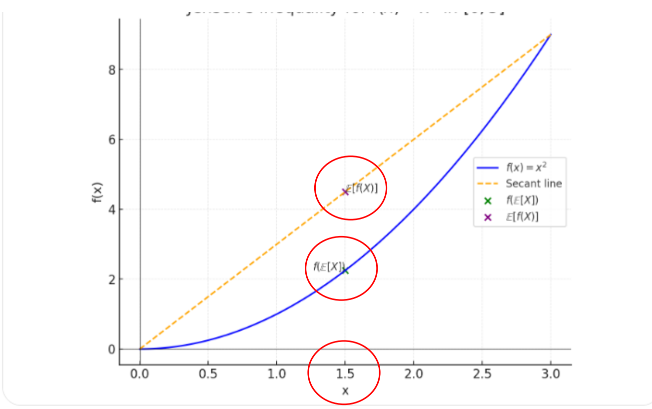
Maximization of $\log p(x)$ can be reduced to the problem of maximizing its ELBO.

Jensen's Inequality

- Jensen's inequality is a fundamental result in mathematics, particularly in convex analysis and probability theory. It states:
  - If f(x) is a **convex function** and X is a random variable, then

    $f(E[X]) \leq E[f(X)]$;
  - If f(x) is a **concave function**, then

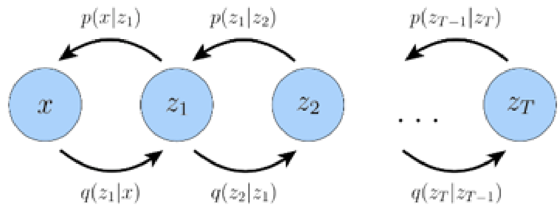
$f(E[X]) \geq E[f(X)]$.



This graph illustrates **Jensen's inequality** for $f(x) = x^2$ in the domain $[0, 3]$:

- The **blue curve** represents the convex function $f(x) = x^2$.
- The **orange dashed line** is the secant line ↓ nnecting $(0, f(0))$ and $(3, f(3))$

Diffusion Model as Markovian Hierarchical VAE (MHVAE)



Generation: $$p(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$$

Encoding: $$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1|\boldsymbol{x}) \prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$$

ELBO of Diffusion Model (replacing z by $z_{1:T}$)

$$\log p(x) = log \int p(x, z_{1:T}) dz_{1:T}$$

$$= log \int \frac{p(x, z_{1:T}) q(z_{1:T}|x)}{q(z_{1:T}|x)} dz_{1:T}$$ (Multiplly by $\frac{q(z|x)}{q(z|x)} = 1$)

$$= log \mathbb{E}_{q(z_{1:T}|x)} \left[\frac{p(x, z_{1:T})}{q(z_{1:T}|x)} \right]$$ (Definition of Expectation)

$$\geq \mathbb{E}_{q(z_{1:T}|x)} \left[log \frac{p(x, z_{1:T})}{q(z_{1:T}|x)} \right]$$ (Apply Jensen's Inequality)

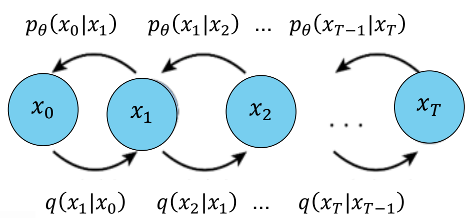
From now on, we change notation as $x \to x_0$, $z_i \to x_i$ and $p(x, z_{1:T}) \to p_\theta(x_{0:T})$.

ELBO of Diffusion Model (notation changed)

$$\begin{aligned}\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}_{0:T}) d\boldsymbol{x}_{1:T} \\ &= \log \int \frac{p(\boldsymbol{x}_{0:T}) q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} d\boldsymbol{x}_{1:T} \\ &= \log \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right] \\ &\geq \underbrace{\mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right]}_{\text{ELBO}}\end{aligned}$$

ELBO of Diffusion Model

[Calvin Luo, 2022]



$$\begin{aligned}\log p(\boldsymbol{x}) &\geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)}\right] \\ &= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) \,||\, p(\boldsymbol{x}_T))}_{\text{prior matching term}} \\ &\quad - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t))\right]}_{\text{denoising matching term}}\end{aligned}$$

(Proof skipped: see the Appendix and the tutorial by Clavin Luo for details)

DIFUSCO as a generic (supervised) CO Solver

- We define a CO problem (e.g., TSP or MIS) as the task of generating discrete {0,1}-vector $x \in \{0,1\}^{N(s)}$, which indicates a sequence (or set) of selected edges or nodes from an input graph, and $N(s)$ is the number of nodes or edges in the input graph.
- We train a neural network to minimize

$$L(\theta) = \mathbb{E}_{s\in S}[-log p_\theta(x_s^*)]$$

where x_s^* is an optimal solution for graph s; $p_\theta(x_s)$ is the system-estimated probability of any $x \in \{0,1\}^{N(s)}$ to be an optimal solution in this graph.

- We obtain a labeled training set for supervised learning of the model, with an existing heuristic (non-ML) solver that finds an (near-)optimal solution for each instance graph.

25

Forward Encoding (Discrete Model)
(a pre-designed procedure, no learning)

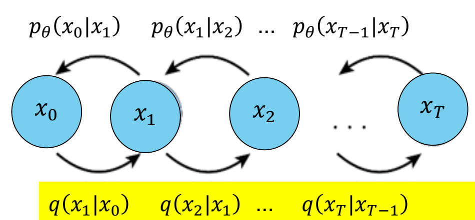
- Manually specify hyperparameters $(\beta_1, \beta_2, \ldots, \beta_T)$, namely the *schedule*, which contains the step-wise corruption ratios.
- Starting with clean data x_0, inject noise in step $t = 1, \ldots, T$ with for each element of x_t as

$$q\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\right) = Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_{t-1}^{(i)} Q_t\right)$$

- Multi-step acceleration (e.g., randomly sample 50 time stamps out of $1, \ldots, T$=1000)

$$q\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, x_0^{(i)}\right) = Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_0^{(i)} \bar{Q}_t\right)$$

$$\bar{Q}_t \triangleq Q_1 Q_2 \ldots Q_t$$



Efficient computation of x_t given x_0 (with closed-form instead of MCMC)

26

Forward Encoding Details (Discrete Model)

- Each step, the element-wise corruption is defined as

$$q\left(x_t^{(i)} \middle| x_{t-1}^{(i)}\right) = Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \boxed{\tilde{x}_{t-1}^{(i)} Q_t}\right)$$

$$p_t^{(i)} = \overbrace{\left(x_{t-1}^{(i)} \quad 1-x_{t-1}^{(i)}\right)}^{\tilde{x}_{t-1}^{(i)}} \overbrace{\begin{pmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{pmatrix}}^{Q_t}$$

$$= \begin{cases} (1-\beta_t,\ \beta_t) & if\, x_{t-1}^{(i)}=1 \\ (\beta_t, 1-\beta_t) & if\ x_{t-1}^{(i)}=0 \end{cases}$$

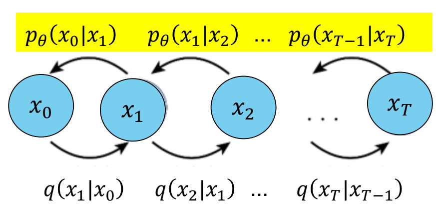
- Applying it recursively yields

$$q\left(x_t^{(i)} \middle| x_{t-1}^{(i)}, x_0^{(i)}\right) = Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_{t-1}^{(i)} Q_t\right)$$
$$= Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_{t-2}^{(i)} Q_{t-1} Q_t\right)$$
$$= Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_{t-3}^{(i)} Q_{t-2} Q_{t-1} Q_t\right)$$
$$\vdots$$
$$= Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_0^{(i)} Q_1 Q_2 \cdots Q_t\right)$$
$$= Cat\left(x_t^{(i)} \middle| p_t^{(i)} = \tilde{x}_0^{(i)} \bar{Q}_t\right) \qquad \bar{Q}_t \triangleq Q_1 Q_2 \ldots Q_t$$

Backward Denoising (Generation)
(supervised learning with a graph neural network)

- Start with a fully noisy x_T whose elements $x_T^{(i)} \sim Ber(0.5)$ are sampled independently.
- For $t = T, \ldots, 1$, calculate

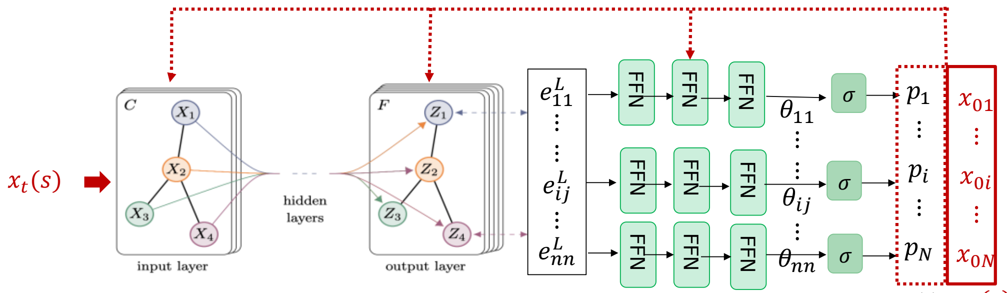
$$p_\theta(x_{t-1}|x_t) = \sum_{\hat{x}_0} q(x_{t-1}|x_t, \hat{x}_0) p_\theta(\hat{x}_0|x_t)$$



where $p_\theta(\hat{x}_0|x_t)$ is estimated by a trained neural network for multi-label classification on labeled set $\mathcal{D} = \{(x_t, x_0)\}_{t=1}^{|\mathcal{D}|}$, which takes x_t as the input and predicts $\hat{x}_0 \in \{0,1\}^N$ with $p_\theta(\hat{x}_0|x_t)$.

Graph-based Denoising Network for $x_t(s) \xrightarrow{mapping} x_0(s)$

Backword propagation with the gradient of $L(\theta) = \mathbb{E}_{s \in S}[-log p_\theta(x_0(s))]$



Anisotropic GNN
(for node & edge embeddings)

MLP

Training the Denoising Network (details)

- Randomly sample K temporal stamps out of the T steps (e.g., K=50, T= 1000);
- Compute x_t given x_0 for $t = 1$ to K using a close-form solution (slide #26);
- Train the GNN+MLP model (last slide) on labeled set $\mathcal{D} = \{(x_t, x_0)\}_{t=1}^{K}$;
- GNN input layers
  - For **TSP**, edge e_{ij}^0 is initialized as the corresponding value in $x_t(s)$, and node h_i^0 is initialized as sinusoidal features (e.g., the geographical coordinates) of the nodes in the input graph;
  - For **MIS**, edge e_{ij}^0 is initialized as zeros, and node h_i^0 is initialized as the corresponding values in $x_t(s)$;

Solution generation per graph in the testing phase

- Start with $x_T \in \{0,1\}^N$ with randomly sampled elements $x_T^{(i)} \sim Ber(0.5)$.
- Use the trained denoising network (GNN) to estimate

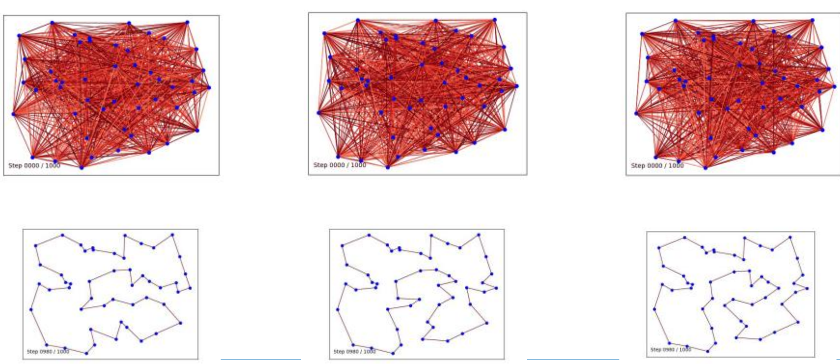
$$p_\theta\left(x_{t-1}^{(i)} \middle| x_t^{(i)}\right) = \Sigma_{\hat{x}_0^i \in \{0,1\}}\, q\left(x_{t-1}^{(i)} \middle| x_t^{(i)}, \hat{x}_0^{(i)}\right) p_\theta\left(\hat{x}_0^{(i)} | x_t^{(i)}\right)$$

  - $q(x_{t-1}|x_t, \hat{x}_0)$ is computed efficiently in a closed form (slide #26);
  - $p_\theta\left(\hat{x}_0^{(i)} | x_t^{(i)}\right)$ is the output at the current step by the GNN.
- Sample $x_{t-1}^{(i)} \sim p_\theta\left(x_{t-1}^{(i)} \middle| x_t^{(i)}\right)$ for $\mathrm{i} = 1, \ldots, N$; set $t \coloneqq t-1$, and repeat above process until $t = 0$.
- Fast inference with multi-step acceleration can be done (e.g., collapsing every 50 steps into one).

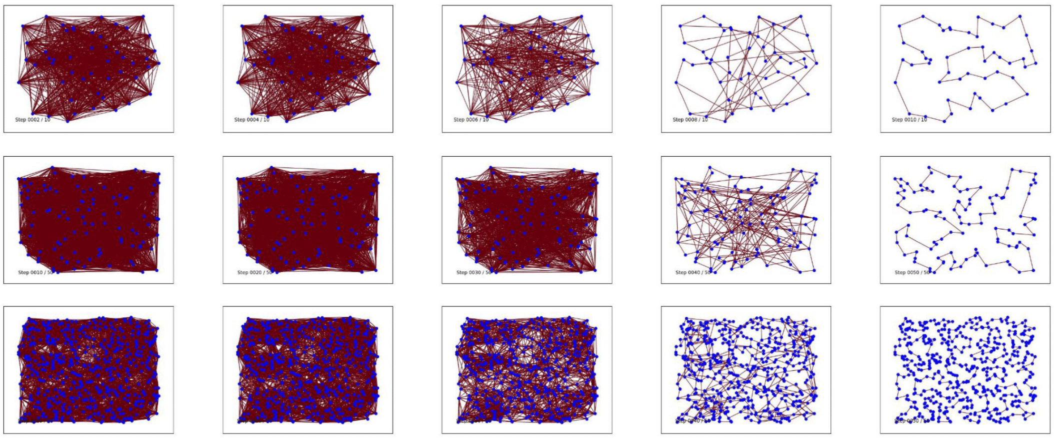
Demo of TSP Solution Generation

(Sun & Yang, NeurIPS 2023)

- Diffusion Model can generate diverse solutions



DIFUSCO on TSP-50, TSP-100, TSP-500



Results on Small Problems [Sun & Yang, NeurIPS 2023]

Algorithm	Type	TSP-50 Length↓	TSP-50 Gap(%)↓	TSP-100 Length↓	TSP-100 Gap(%)↓
Concorde*	Exact	5.69	0.00	7.76	0.00
2-OPT	Heuristics	5.86	2.95	8.03	3.54
AM	Greedy	5.80	1.76	8.12	4.53
GCN	Greedy	5.87	3.10	8.41	8.38
Transformer	Greedy	5.71	0.31	7.88	1.42
POMO	Greedy	5.73	0.64	7.84	1.07
Sym-NCO	Greedy	-	-	7.84	0.94
DPDP	$1k$-Improvements	5.70	0.14	7.89	1.62
Image Diffusion	Greedy†	5.76	1.23	7.92	2.11
Ours	Greedy†	**5.70**	**0.10**	**7.78**	**0.24**
AM	$1k\times$Sampling	5.73	0.52	7.94	2.26
GCN	$2k\times$Sampling	5.70	0.01	7.87	1.39
Transformer	$2k\times$Sampling	5.69	0.00	7.76	0.39
POMO	$8\times$Augment	5.69	0.03	7.77	0.14
Sym-NCO	$100\times$Sampling	-	-	7.79	0.39
MDAM	$50\times$Sampling	5.70	0.03	7.79	0.38
DPDP	$100k$-Improvements	5.70	0.00	7.77	0.00
Ours	$16\times$Sampling	**5.69**	**-0.01**	**7.76**	**-0.01**

*Concorde is a TSP solver for integer coordinates.

Results on Large Problems [Sun & Yang, NeurIPS 2023]

Algorithm	Type	TSP-500			TSP-1000			TSP-10000		
		Length ↓	Gap ↓	Time ↓	Length ↓	Gap ↓	Time ↓	Length ↓	Gap ↓	Time ↓
Concorde	Exact	16.55*	—	37.66m	23.12*	—	6.65h	N/A	N/A	N/A
Gurobi	Exact	16.55	0.00%	45.63h	N/A	N/A	N/A	N/A	N/A	N/A
LKH-3 (default)	Heuristics	16.55	0.00%	46.28m	23.12	0.00%	2.57h	71.77*	—	8.8h
LKH-3 (less trails)	Heuristics	16.55	0.00%	3.03m	23.12	0.00%	7.73m	71.79	—	51.27m
Farthest Insertion	Heuristics	18.30	10.57%	0s	25.72	11.25%	0s	80.59	12.29%	6s
AM	RL+G	20.02	20.99%	1.51m	31.15	34.75%	3.18m	141.68	97.39%	5.99m
GCN	SL+G	29.72	79.61%	6.67m	48.62	110.29%	28.52m	N/A	N/A	N/A
POMO+EAS-Emb	RL+AS+G	19.24	16.25%	12.80h	N/A	N/A	N/A	N/A	N/A	N/A
POMO+EAS-Tab	RL+AS+G	24.54	48.22%	11.61h	49.56	114.36%	63.45h	N/A	N/A	N/A
DIMES	RL+G	18.93	14.38%	0.97m	26.58	14.97%	2.08m	86.44	20.44%	4.65m
DIMES	RL+AS+G	17.81	7.61%	2.10h	24.91	7.74%	4.49h	80.45	12.09%	3.07h
Ours (DIFUSCO)	SL+G†	18.35	10.85%	3.61m	26.14	13.06%	11.86m	98.15	36.75%	28.51m
Ours (DIFUSCO)	SL+G†+2-opt	**16.80**	**1.49%**	**3.65**m	**23.56**	**1.90%**	**12.06**m	**73.99**	**3.10%**	**35.38**m
EAN	RL+S+2-opt	23.75	43.57%	57.76m	47.73	106.46%	5.39h	N/A	N/A	N/A
AM	RL+BS	19.53	18.03%	21.99m	29.90	29.23%	1.64h	129.40	80.28%	1.81h
GCN	SL+BS	30.37	83.55%	38.02m	51.26	121.73%	51.67m	N/A	N/A	N/A
DIMES	RL+S	18.84	13.84%	1.06m	26.36	14.01%	2.38m	85.75	19.48%	4.80m
DIMES	RL+AS+S	17.80	7.55%	2.11h	24.89	7.70%	4.53h	80.42	12.05%	3.12h
Ours (DIFUSCO)	SL+S	17.23	4.08%	11.02m	25.19	8.95%	46.08m	95.52	33.09%	6.59h
Ours (DIFUSCO)	SL+S+2-opt	**16.65**	**0.57%**	**11.46**m	**23.45**	**1.43%**	**48.09**m	**73.89**	**2.95%**	**6.72**h
Att-GCN	SL+MCTS	16.97	2.54%	2.20m	23.86	3.22%	4.10m	74.93	4.39%	21.49m
DIMES	RL+MCTS	16.87	1.93%	2.92m	23.73	2.64%	6.87m	74.63	3.98%	29.83m
DIMES	RL+AS+MCTS	16.84	1.76%	2.15h	23.69	2.46%	4.62h	74.06	3.19%	3.57h
Ours (DIFUSCO)	SL+MCTS	**16.63**	**0.46%**	**10.13**m	**23.39**	**1.17%**	**24.47**m	**73.62**	**2.58%**	**47.36**m

Concluding Remarks

- ~~**Bad News**: ML solvers cannot beat exact solvers for small graphs ($n \le 100$ nodes) because exact solvers guarantees to find optimal solutions when they can scale.~~
- **Good News**: ML solvers, especially with the advanced NAR neural solvers (e.g., DIMES and DIFUSCO), can fund near-optimal solutions for large problems that traditional exact solvers fail to scale up.
- **Good News**: The advanced NAR neural solvers have achieved the performance level close to the best domain-specific heuristic solvers but without hand-craft heuristics.
- **Follow-up efforts**: training diffusion-based models in unsupervised setting (without knowing the best solution per graph); pretrained LLMs (next)

References

- [NeurIPS 2022] R Qiu*, Z Sun*, Z. and Y Yang. DIMES: A differentiable meta solver for combinatorial optimization problems
- [NeurIPS 2023] Sun, Z. and Yang, Y. Difusco: Graph-based Diffusion Solvers for Combinatorial Optimization
- [Tutorial 2022] Calvin Luo. Understanding Diffusion Models: A Unified Perspective: An intuitive, accessible tutorial on diffusion models.

40

Appendix: ELBO of the Diffusion Model

$$
\begin{aligned}
\log p(\boldsymbol{x}) &\geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)\prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)\prod_{t=2}^{T} \boxed{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})}}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)\prod_{t=2}^{T} \boxed{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0)}}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} + \log \prod_{t=2}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\boxed{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0)}}\right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} + \log \prod_{t=2}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\boxed{\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)q(\boldsymbol{x}_t|\boldsymbol{x}_0)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}}}\right]
\end{aligned}
$$

Due to Markov property

Due to Bayes rule

41

Appendix: ELBO of the Diffusion Model

$$\mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} + \log \prod_{t=2}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)q(\boldsymbol{x}_t|\boldsymbol{x}_0)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}}\right]$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} + \log \prod_{t=2}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)\cancel{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}}{\cancel{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}}}\right]$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{\cancel{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}} + \log \frac{\cancel{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}}{q(\boldsymbol{x}_T|\boldsymbol{x}_0)} + \log \prod_{t=2}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)}\right]$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_T|\boldsymbol{x}_0)} + \sum_{t=2}^{T} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)}\right]$$

$$= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)}\right]$$

$$= \mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right] + \mathbb{E}_{q(\boldsymbol{x}_T|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_0)}\right] + \sum_{t=2}^{T} \mathbb{E}_{q(\boldsymbol{x}_t,\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)}\right]$$

$$= \boxed{\underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]}_{\text{reconstruction term}}} - \boxed{\underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) \parallel p(\boldsymbol{x}_T))}_{\text{prior matching term}}} - \boxed{\sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]}_{\text{denoising matching term}}}$$