Stanford MS&E/CS 331:
Diffusion models for neural combinatorial optimization

Ellen Vitercik*

October 28, 2025

These lecture notes provide a short introduction to discrete diffusion models. We ground the discussion in the context of the paper DIFUSCO by Sun and Yang [2], which uses diffusion to find high-quality solutions to combinatorial problems such as the traveling salesman problem (TSP) and maximum independent set.

1 Motivation: Probabilistic neural combinatorial optimization

To ground the motivation in probabilistic neural combinatorial optimization (NCO), we assume sample access to a distribution $\mathcal{D}$ over combinatorial problem instances s, such as Erdős–Rényi graphs in graph-based optimization tasks. For each instance s, the space of candidate solutions is denoted by $\mathcal{X}_s = \{0,1\}^N$, representing binary decision vectors such as edge selections in a TSP tour.

Probabilistic NCO solvers define a parameterized distribution $q_{\boldsymbol{\theta}}(\boldsymbol{x} \mid s)$ over candidate solutions, where $\boldsymbol{\theta}$ denotes the trainable parameters of a neural network. The goal is to learn $p_{\boldsymbol{\theta}}$ so that it assigns high probability to high-quality solutions for each problem instance.

To train such a model, we draw a finite training set of problem instances $\mathcal{S} \sim \mathcal{D}^n$, each labeled with an optimal solution $\boldsymbol{x}^{s*}$. The learning objective is to maximize the likelihood of these labeled solutions, or equivalently, to minimize the negative log-likelihood:

$$L(\boldsymbol{\theta}) = \sum_{s \in \mathcal{S}} -\log q_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{s*} \mid s\right).$$

The diffusion-based approach frames this process as learning to reverse a structured corruption procedure. Each ground-truth solution $\boldsymbol{x}^{s*}$ is progressively corrupted through a multi-step noise process (the *forward encoder*) that eventually transforms it into a sample drawn from the uniform distribution over $\{0,1\}^N$. A neural network is then trained to invert this corruption, step by step, effectively denoising the data to recover the clean solution (the *reverse decoder*). At inference time, the model begins from uniform noise and iteratively denoises it to generate high-quality solutions for new problem instances.

2 Forward encoder

Given an optimal solution $\boldsymbol{x}_0 = \boldsymbol{x}^{s*}$, the forward diffusion process begins by introducing random corruption to obtain a noised version $\boldsymbol{x}_1$. Each bit of $\boldsymbol{x}_0$ is independently flipped with probability β_1 and remains unchanged with probability $1 - \beta_1$.

*These notes are course material and have not undergone formal peer review. Please feel free to send me any typos or comments.

To formalize this process, we first represent $\boldsymbol{x}_0 \in \{0,1\}^N$ as a one-hot matrix $\tilde{\boldsymbol{x}}_0 \in \{0,1\}^{N\times 2}$, where each row encodes whether the corresponding bit is 0 or 1:

$$\tilde{\boldsymbol{x}}_0[i,:] = \begin{cases} [1,0], & \text{if } x_0[i] = 0, \\ [0,1], & \text{if } x_0[i] = 1. \end{cases}$$

We then define the transition probability matrix

$$Q_1 = \begin{pmatrix} 1-\beta_1 & \beta_1 \\ \beta_1 & 1-\beta_1 \end{pmatrix},$$

which specifies the probability of either retaining or flipping each bit during the corruption step.

Multiplying the one-hot representation by this matrix yields the conditional probabilities for each bit after the first noising step:

$$\boldsymbol{p} = \tilde{\boldsymbol{x}}_0 Q_1, \qquad \boldsymbol{p}[i,:] = \begin{cases} [1-\beta_1,\ \beta_1], & \text{if } x_0[i] = 0, \\ [\beta_1,\ 1-\beta_1], & \text{if } x_0[i] = 1. \end{cases}$$

Finally, the corrupted sample $\boldsymbol{x}_1$ is drawn from a categorical distribution defined by these probabilities: $\boldsymbol{x}_1 \sim q(\boldsymbol{x}_1 \mid \boldsymbol{x}_0, s) = \mathrm{Cat}(\boldsymbol{x}_1; \boldsymbol{p} = \tilde{\boldsymbol{x}}_0 Q_1)$, meaning that for each index i,

$$x_1[i] = \begin{cases} 0, & \text{with probability } p[i,0], \\ 1, & \text{with probability } p[i,1]. \end{cases}$$

This step constitutes the first stage of the forward (noising) process, which then proceeds iteratively for timesteps $t = 2, 3, \ldots, T$. At each step, we begin with a binary vector $\boldsymbol{x}_{t-1} \in \{0,1\}^N$ and construct its one-hot representation $\tilde{\boldsymbol{x}}_{t-1} \in \{0,1\}^{N\times 2}$. A timestep-dependent transition matrix $Q_t \in \mathbb{R}^{2\times 2}$ is then defined using the scalar noise parameter β_t:

$$Q_t = \begin{pmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{pmatrix}, \qquad \text{where } \beta_1 > \beta_2 > \cdots > \beta_T.$$

Each bit of $\boldsymbol{x}_{t-1}$ is then randomly flipped according to this matrix by sampling

$$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, s) = \mathrm{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \tilde{\boldsymbol{x}}_{t-1} Q_t). \tag{1}$$

Thus, Q_t controls the level of corruption introduced at each timestep, with larger β_t values corresponding to higher noise.

By composing the transitions across multiple steps, the marginal distribution of the noised variable can be expressed as

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_0, s) = q(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s) = \mathrm{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \tilde{\boldsymbol{x}}_0 \bar{Q}_t) \tag{2}$$

where $\bar{Q}_t = Q_1 Q_2 \cdots Q_t$. The sequence $\beta_1, \ldots, \beta_T$ is chosen such that

$$\prod_{t=1}^{T} (1-\beta_t) \approx 0,$$

ensuring that after sufficiently many timesteps, the distribution $q(\boldsymbol{x}_T \mid \boldsymbol{x}^{s*}, s)$ approaches the uniform distribution over $\{0,1\}^N$. This property guarantees that the forward process gradually destroys information in the original solution until only pure noise remains, providing the foundation for learning its reverse denoising process.

3 Reverse decoder

The objective of the reverse diffusion process is to learn how to invert the noise introduced during the forward process. Formally, we aim to model the conditional distribution $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s)$, which describes how a partially denoised solution $\boldsymbol{x}_{t-1}$ can be sampled given the current noisy state $\boldsymbol{x}_t$ and the problem instance s.

To derive this quantity, we first consider the distribution conditioned on both the problem instance and its corresponding optimal solution: $(s, \boldsymbol{x}^{s*})$. Applying Bayes' rule gives:

$$q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s\right) = \frac{q\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}^{s*}, s\right) q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}^{s*}, s\right)}{q\left(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s\right)}.$$

Note that each state $\boldsymbol{x}_t$ depends only on its immediate predecessor $\boldsymbol{x}_{t-1}$ and not on earlier states given $\boldsymbol{x}_{t-1}$, so the expression simplifies to:

$$q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s\right) = \frac{q\left(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}\right) q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}^{s*}, s\right)}{q\left(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s\right)}. \tag{3}$$

Here, the denominator $q\left(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s\right)$ acts as a normalization constant since it does not depend on $\boldsymbol{x}_{t-1}$.

Squinting our eyes and pattern matching with Equations (1) and (2), we can write this conditional distribution explicitly:

$$q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s\right) = \mathrm{Cat}\left(\boldsymbol{x}_{t-1}; \boldsymbol{p} = \frac{\tilde{\boldsymbol{x}}_t Q_t^\top \odot \tilde{\boldsymbol{x}}_0 \bar{Q}_{t-1}}{\tilde{\boldsymbol{x}}_0 \bar{Q}_t \tilde{\boldsymbol{x}}_t^\top}\right), \tag{4}$$

where $\odot$ denotes element-wise multiplication. (For a full derivation, see the online notes by Beckham [1].)

This expression gives the exact posterior distribution over $\boldsymbol{x}_{t-1}$ given the current state $\boldsymbol{x}_t$ and the clean data $\boldsymbol{x}^{s*}$. However, during inference we do not have access to the true clean solution $\boldsymbol{x}^{s*}$, so we must marginalize over all possible ground-truth solutions:

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s) = \sum_{\boldsymbol{x}^{s*}} q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s\right) q\left(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s\right) = \mathbb{E}_{\boldsymbol{x}^{s*} \sim q(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)}\left[q\left(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s\right)\right].$$

Of course, at test time, we do not have access to the true posterior distribution $q(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)$. Instead, we approximate it using a learned model $p_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)$, where $\boldsymbol{\theta}$ are the parameters of a neural network. This leads to the following approximation:

$$q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s) \approx \mathbb{E}_{\boldsymbol{x}^{s*} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)}[q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}^{s*}, s)] := q_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s).$$

To evaluate this expectation without summing over an exponentially large space of possible solutions, we apply Bayes' rule and the Markov property of the diffusion process:

$$\begin{aligned}
q_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s) &= \mathbb{E}_{\boldsymbol{x}^{s*} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)}\left[\frac{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}^{s*}, s)\, q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}^{s*}, s)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s)}\right] \\
&= \mathbb{E}_{\boldsymbol{x}^{s*} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)}\left[\frac{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})\, q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}^{s*}, s)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s)}\right] \\
&= q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})\, \mathbb{E}_{\boldsymbol{x}^{s*} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_t, s)}\left[\frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}^{s*}, s)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}^{s*}, s)}\right].
\end{aligned}$$

Then, this expectation is approximated using a single Monte Carlo sample. In particular, we draw one sample $\boldsymbol{x}_0 \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_t, s)$ and compute:

$$q_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s) \approx q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) \cdot \frac{q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_0, s)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_0, s)}.$$

Substituting the expressions for the categorical distributions from Equations (3) and (4), we obtain:

$$q_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, s) = \mathrm{Cat}\left(\boldsymbol{x}_{t-1};\, \boldsymbol{p} = \frac{\tilde{\boldsymbol{x}}_t Q_t^\top \odot \tilde{\boldsymbol{x}}_0 \bar{Q}_{t-1}}{\tilde{\boldsymbol{x}}_0 \bar{Q}_t \tilde{\boldsymbol{x}}_t^\top}\right).$$

This learned approximation enables the model to sample plausible denoised configurations at each timestep without requiring access to the true optimal solution $\boldsymbol{x}^{s*}$.

4 Training the decoder

The goal of the decoder is to reconstruct the clean solution $\boldsymbol{x}^{s*}$ from a noisy input $(\boldsymbol{x}_t, s)$, where $\boldsymbol{x}_t$ represents a corrupted version of a problem instance s. In the context of specific combinatorial optimization problems, this prediction task takes on different interpretations:

- **TSP:** The target $\boldsymbol{x}^{s*}$ indicates whether each edge in the graph is part of the optimal tour.
- **Maximum independent set (MIS):** The target $\boldsymbol{x}^{s*}$ identifies which vertices belong to the largest subset of mutually non-adjacent nodes. In this case, each entry of $\boldsymbol{x}^{s*}$ specifies whether a given vertex is included in the independent set.

Sun and Yang [2] implement the decoder as an *anisotropic graph neural network (AGNN)*, which performs *directed* message passing along the edges of the graph. Each edge aggregates information from its incident vertices and propagates messages that capture local structure and directionality. The initialization of node and edge embeddings depends on the problem type:

- **TSP:** Sun and Yang [2] study TSP when the city locations are embedded in two-dimensional Euclidean space. Edge features include the pairwise Euclidean distances between nodes and their corresponding labels from $\boldsymbol{x}_t$. Node features consist of the 2D coordinates of the cities.
- **MIS:** Node features are initialized using their labels in $\boldsymbol{x}_t$ and the edges do not have initial embeddings.

In both cases, the denoising timestep t is encoded using a sinusoidal positional embedding, which allows the model to modulate its behavior based on the current noise level. The decoder is trained using a cross-entropy loss.

5 Decoding Strategies

At test time, after completing T denoising steps on a problem instance s, the model produces a final distribution $q_{\boldsymbol{\theta}}(\boldsymbol{x}^{s*} \mid \boldsymbol{x}_1, s)$ over candidate solutions. Individual samples drawn from this distribution are not guaranteed to be feasible solutions to the underlying combinatorial problem. Consequently, task-specific decoding heuristics are applied to convert the probabilistic outputs into valid solutions.

TSP. Let $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$ denote the Euclidean coordinates of cities i and j. The diffusion model assigns a probability mass to each edge (i, j), indicating the likelihood that it is part of the optimal tour. To generate a feasible tour:

- Edges are ranked according to the ratio of their predicted probability mass to their Euclidean distance, i.e.,
$$\frac{\text{probability mass on } (i,j)}{\|\boldsymbol{c}_i - \boldsymbol{c}_j\|}.$$
- A **greedy decoding** algorithm sequentially inserts edges in this ranked order while avoiding conflicts such as premature cycles or degree violations.
- A few rounds of **2-opt local search** are optionally applied to refine the resulting tour by iteratively swapping edges to reduce the total travel distance.

MIS. For MIS, decoding proceeds analogously: nodes are ranked according to their predicted probability mass, and a **greedy selection** process is used to add vertices to the independent set as long as doing so does not introduce adjacency conflicts.

Sampling decoding. To further enhance solution quality, multiple decoding runs can be performed in parallel, each initialized with a different random noise sample $\boldsymbol{x}_T$. This **sampling decoding** strategy allows the model to explore distinct regions of the solution space and select the best outcome according to the problem's objective function.

6 Experiments

6.1 TSP

Each TSP instance consists of a set of cities uniformly sampled from the unit square, with edge weights corresponding to Euclidean distances. To reduce computational complexity on large graphs, each node is connected only to its k nearest neighbors (50 for TSP-500 and 100 for TSP-1000 and TSP-10000). During training, $T = 1000$ diffusion steps are used. During testing, two settings are compared: a 50-step 1-sample policy for greedy decoding, and a 10-step 16-sample policy for sampling decoding. Figure 1 (a screenshot of Table 2 by Sun and Yang [2]) presents the main comparison against state-of-the-art neural and heuristic baselines.

6.2 MIS

The MIS experiments are on two classes of graphs: SATLIB graphs, derived from Boolean satisfiability (SAT) instances, and Erdős–Rényi (ER) random graphs. In the ER-[700–800] benchmark, each graph contains between 700 and 800 nodes, with a connection probability of 0.15. Training is again performed with $T = 1000$ diffusion steps. For SATLIB graphs, the authors use two inference modes: a 50-step 1-sample policy for greedy decoding and a 50-step 4-sample policy for sampling decoding. For ER graphs, they use a 50-step 1-sample policy for greedy decoding and a 20-step 8-sample policy for sampling decoding. Figure 1 (a screenshot of Table 3 by Sun and Yang [2]) presents the main comparison against state-of-the-art neural and heuristic baselines.

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

Figure 1: TSP results by Sun and Yang [2].

Method	Type	SATLIB			ER-[700-800]		
		Size ↑	Gap ↓	Time ↓	Size ↑	Gap ↓	Time ↓
KaMIS	Heuristics	425.96*	—	37.58m	44.87*	—	52.13m
Gurobi	Exact	425.95	0.00%	26.00m	41.38	7.78%	50.00m
Intel	SL+G	420.66	1.48%	23.05m	34.86	22.31%	6.06m
Intel	SL+TS	N/A	N/A	N/A	38.80	13.43%	20.00m
DGL	SL+TS	N/A	N/A	N/A	37.26	16.96%	22.71m
LwD	RL+S	422.22	0.88%	18.83m	41.17	8.25%	6.33m
DIMES	RL+G	421.24	1.11%	24.17m	38.24	14.78%	6.12m
DIMES	RL+S	423.28	0.63%	20.26m	**42.06**	**6.26%**	12.01m
Ours	SL+G	**424.50**	**0.34%**	8.76m	38.83	12.40%	8.80m
Ours	SL+S	**425.13**	**0.21%**	23.74m	41.12	8.36%	26.67m

Figure 2: MIS results by Sun and Yang [2].

References

[1] Christopher Beckham. My notes on discrete denoising diffusion models (D3PMs). https://beckham.nz/2022/07/11/d3pms.html, April 2023. Accessed: 2025-10-27.

[2] Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.