SymmetricDiffusers: Learning Discrete Diffusion on Finite Symmetric Groups

Anonymous Author(s)
Affiliation
Address
email

Abstract

Finite symmetric groups S_n are essential in fields such as combinatorics, physics, and chemistry. However, learning a probability distribution over S_n poses significant challenges due to its intractable size and discrete nature. In this paper, we introduce SymmetricDiffusers, a novel discrete diffusion model that simplifies the task of learning a complicated distribution over S_n by decomposing it into learning simpler transitions of the reverse diffusion using deep neural networks. We identify the riffle shuffle as an effective forward transition and provide empirical guidelines for selecting the diffusion length based on the theory of random walks on finite groups. Additionally, we propose a generalized Plackett-Luce (PL) distribution for the reverse transition, which is provably more expressive than the PL distribution. We further introduce a theoretically grounded "denoising schedule" to improve sampling and learning efficiency. Extensive experiments show that our model achieves state-of-the-art or comparable performances on solving tasks including sorting 4-digit MNIST images, jigsaw puzzles, and traveling salesman problems.

1 Introduction

As a vital area of abstract algebra, finite groups provide a structured framework for analyzing symmetries and transformations which are fundamental to a wide range of fields, including combinatorics, physics, chemistry, and computer science. One of the most important finite groups is the *finite symmetric group* S_n, defined as the group whose elements are all the bijections (or permutations) from a set of n elements to itself, with the group operation being function composition.

Classic probabilistic models for finite symmetric groups S_n, such as the Plackett-Luce (PL) model [35, 27], the Mallows model [28], and card shuffling methods [9], are crucial in analyzing preference data and understanding the convergence of random walks. Therefore, studying probabilistic models over S_n through the lens of modern machine learning is both natural and beneficial. This problem is theoretically intriguing as it bridges abstract algebra and machine learning. For instance, Cayley's Theorem, a fundamental result in abstract algebra, states that every group is isomorphic to a subgroup of a symmetric group. This implies that learning a probability distribution over finite symmetric groups could, in principle, yield a distribution over any finite group. Moreover, exploring this problem could lead to the development of advanced models capable of addressing tasks such as permutations in ranking problems, sequence alignment in bioinformatics, and sorting.

However, learning a probability distribution over finite symmetric groups S_n poses significant challenges. First, the number of permutations of n objects grows factorially with n, making the inference and learning computationally expensive for large n. Second, the discrete nature of the data brings difficulties in designing expressive parameterizations and impedes the gradient-based learning.

In this work, we propose a novel discrete (state space) diffusion model over finite symmetric groups, dubbed as *SymmetricDiffusers*. It overcomes the above challenges by decomposing the difficult

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

problem of learning a complicated distribution over S_n into a sequence of simpler problems, *i.e.*, learning individual transitions of a reverse diffusion process using deep neural networks. Based on the theory of random walks on finite groups, we investigate various shuffling methods as the forward process and identify the riffle shuffle as the most effective. We also provide empirical guidelines on choosing the diffusion length based on the mixing time of the riffle shuffle. Furthermore, we examine potential transitions for the reverse diffusion, such as inverse shuffling methods and the PL distribution, and introduce a novel generalized PL distribution. We prove that our generalized PL is more expressive than the PL distribution. Additionally, we propose a theoretically grounded "denoising schedule" that merges reverse steps to improve the efficiency of sampling and learning. To validate the effectiveness of our SymmetricDiffusers, we conduct extensive experiments on three tasks: sorting 4-Digit MNIST images, solving Jigsaw Puzzles on the Noisy MNIST and CIFAR-10 datasets, and addressing traveling salesman problems (TSPs). Our model achieves the state-of-the-art or comparable performance across all tasks.

2 Related Works

Random Walks on Finite Groups. The field of random walks on finite groups, especially finite symmetric groups, have been extensively studied by previous mathematicians [37, 11, 4, 38]. Techniques from a variety of different fields, including probability, combinatorics, and representation theory, have been used to study random walks on finite groups [38]. In particular, random walks on finite symmetric groups are first studied in the application of card shuffling, with many profound theoretical results of shuffling established. A famous result in the field shows that 7 riffle shuffles are enough to mix up a deck of 52 cards [4], where a riffle shuffle is a mathematically precise model that simulates how people shuffle cards in real life. The idea of shuffling to mix up a deck of cards aligns naturally with the idea of diffusion, and we seek to fuse the modern techniques of diffusion models with the classical theories of random walks on finite groups.

Diffusion Models. Diffusion models [40, 41, 16, 42] are a powerful class of generative models that typically deals with continuous data. They consist of forward and reverse processes. The forward process is typically a discrete-time continuous-state Markov chain or a continuous-time continuous-state Markov process that gradually adds noise to data, and the reverse process learn neural networks to denoise. Discrete (state space) diffusion models have also been proposed to handle discrete data like image, text [3], and graphs [45]. Existing discrete diffusion models are applicable for learning distributions of permutations. However, they focused on cases where the state space is small or has a special (*e.g.*, decomposable) structure and are unable to deal with intractable-sized state spaces like the symmetric group. In particular, [3] requires an explicit transition matrix, which has size $n! \times n!$ in the case of finite symmetric groups and has no simple representations or sparsifications.

Differentiable Sorting and Learning Permutations. A popular paradigm to learn permutations is through differentiable sorting or matching algorithms. Various differentiable sorting algorithms have been proposed that uses continuous relaxations of permutation matrices [13, 8, 5], or uses differentiable swap functions [33, 34, 20]. The Gumbel-Sinkhorn method [29] has also been proposed to learn latent permutations using the continuous Sinkhorn operator. Such methods often focus on finding the optimal permutation instead of learning a distribution over the finite symmetric group. Moreover, they tend to be less effective as n grows larger due to their high complexities.

3 Learning Diffusion Models on Finite Symmetric Groups

We first introduce some notations. Fix $n \in \mathbb{N}$. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. A *permutation* σ on $[n]$ is a function from $[n]$ to $[n]$, and we usually write σ as $\begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}$. The *identity permutation*, denoted by Id, is the permutation given by $\mathrm{Id}(i) = i$ for all $i \in [n]$. Let S_n be the set of all permutations (or bijections) from a set of n elements to itself, called the ***finite symmetric group***, whose group operation is the function composition. For a permutation $\sigma \in S_n$, the permutation matrix $Q_\sigma \in \mathbb{R}^{n \times n}$ associated with σ satisfies $e_i^\top Q_\sigma = e_{\sigma(i)}^\top$ for all $i \in [n]$. In this paper, we consider a set of n distinctive objects $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where the i-th object is represented by a d-dimensional vector $\mathbf{x}_i$. Therefore, a ranked list of objects can be represented as a matrix $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$, where the ordering of rows corresponds to the ordering of objects. We can permute X via permutation σ to obtain $Q_\sigma X$.

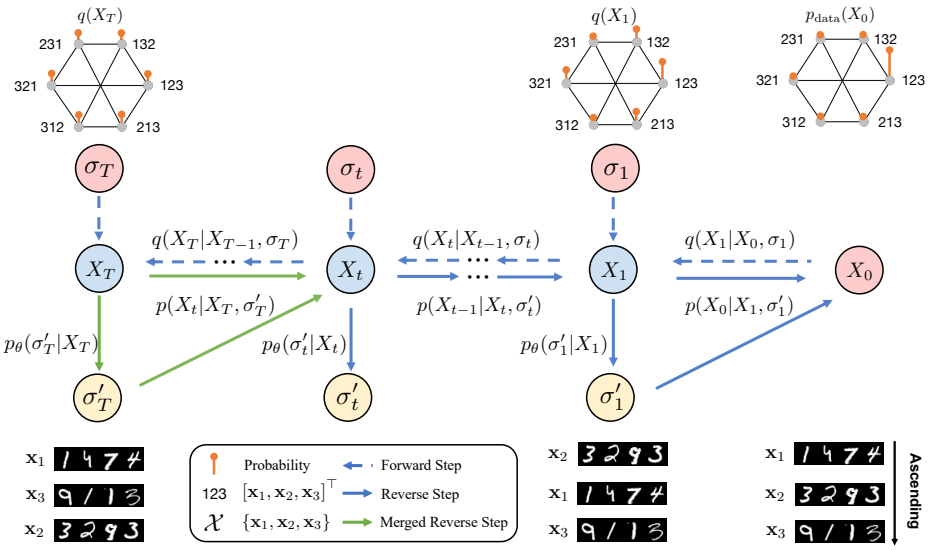


Figure 1: This figure illustrates our discrete diffusion model on finite symmetric groups. The middle graphical model displays the forward and reverse diffusion processes. We demonstrate learning distributions over the symmetric group S_3 via the task of sorting three MNIST 4-digit images. The top part of the figure shows the marginal distribution of a ranked list of images X_t at time t, while the bottom shows a randomly drawn list of images.

Our goal is to learn a distribution over S_n. We propose learning discrete (state space) diffusion models, which consist of a *forward process* and a *reverse process*. In the forward process, starting from the unknown data distribution, we simulate a random walk until it reaches a known stationary "noise" distribution. In the reverse process, starting from the known noise distribution, we simulate another random walk, where the transition probability is computed using a neural network, until it recovers the data distribution. Learning a transition distribution over S_n is often more manageable than learning the original distribution because: (1) the support size (the number of states that can be reached in one transition) could be much smaller than $n!$, and (2) the distance between the initial and target distributions is smaller. By doing so, we break down the hard problem (learning the original distribution) into a sequence of simpler subproblems (learning the transition distribution). The overall framework is illustrated in Fig. 1. In the following, we will introduce the forward card shuffling process in Section 3.1, the reverse process in Section 3.2, the network architecture and training in Section 3.3, denoising schedule in Section 3.4, and reverse decoding methods in Section 3.5.

3.1 Forward Diffusion Process: Card Shuffling

Suppose we observe a set of objects $\mathcal{X}$ and their ranked list X_0. They are assumed to be generated from an unknown data distribution in an IID manner, *i.e.*, $X_0, \mathcal{X} \overset{\text{iid}}{\sim} p_{\text{data}}(X, \mathcal{X})$. One can construct a bijection between a ranked list of n objects and an ordered deck of n cards. Therefore, permuting objects is equivalent to shuffling cards. In the forward diffusion process, we would like to add "random noise" to the rank list so that it reaches to some known stationary distribution like the uniform. Formally, we let $\mathcal{S} \subseteq S_n$ be a set of permutations that are realizable by a given shuffling method in one step. $\mathcal{S}$ does not change across steps in common shuffling methods. We will provide concrete examples later. We then define the *forward process* as a Markov chain,

$$q(X_{1:T}|X_0,\mathcal{X}) = q(X_{1:T}|X_0) = \prod_{t=1}^{T} q(X_t|X_{t-1}), \tag{1}$$

where $q(X_t|X_{t-1}) = \sum_{\sigma_t\in\mathcal{S}} q(X_t|X_{t-1},\sigma_t)q(\sigma_t)$ and the first equality in Eq. (1) holds since X_0 implies $\mathcal{X}$. In the forward process, although the set $\mathcal{X}$ does not change, the rank list of objects X_t changes. Here $q(\sigma_t)$ has the support $\mathcal{S}$ and describes the permutation generated by the underlying shuffling method. Note that common shuffling methods are time-homogeneous Markov chains, *i.e.*, $q(\sigma_t)$ stays the same across time. $q(X_t|X_{t-1},\sigma_t)$ is a delta distribution $\delta\left(X_t = Q_{\sigma_t}X_{t-1}\right)$ since the permuted objects X_t are uniquely determined given the permutation σ_t and X_{t-1}. We denote the *neighbouring states* of X via one-step shuffling as $N_{\mathcal{S}}(X) := \{Q_\sigma X|\sigma\in\mathcal{S}\}$. Therefore, we have,

$$q(X_t|X_{t-1}) = \begin{cases} q(\sigma_t) & \text{if } X_t \in N_{\mathcal{S}}(X_{t-1}) \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that $X_t \in N_{\mathcal{S}}(X_{t-1})$ is equivalent to $\sigma_t \in \mathcal{S}$ and $X_t = Q_{\sigma_t} X_{t-1}$.

3.1.1 Card Shuffling Methods

We now consider several popular shuffling methods as the forward transition, *i.e.*, *random transpositions*, *random insertions*, and *riffle shuffles*. Different shuffling methods provide different design choices of $q(\sigma_t)$, thus corresponding to different forward diffusion processes. Although all these forward diffusion processes share the same stationary distribution, *i.e.*, the uniform, they differ in their mixing time. We will introduce stronger quantitative results on their mixing time later.

Random Transpositions. One natural way of shuffling is to swap pairs of objects. Formally, a *transposition* or a *swap* is a permutation $\sigma \in S_n$ such that there exist $i \neq j \in [n]$ with $\sigma(i) = j$, $\sigma(j) = i$, and $\sigma(k) = k$ for all $k \notin \{i, j\}$, in which case we denote $\sigma = (i \quad j)$. We let $\mathcal{S} = \{(i \quad j) : i \neq j \in [n]\} \cup \{\mathrm{Id}\}$. For any time t, we define $q(\sigma_t)$ by choosing two indices from $[n]$ uniformly and independently and swap the two indices. If the two chosen indices are the same, then this means that we have sampled the identity permutation. Specifically, $q(\sigma_t = (i \quad j)) = 2/n^2$ when $i \neq j$ and $q(\sigma_t = \mathrm{Id}) = 1/n$.

Random Insertions. Another shuffling method is to insert the last piece to somewhere in the middle. Let $\texttt{insert}_i$ denote the permutation that inserts the last piece right before the i^{th} piece, and let $\mathcal{S} := \{\texttt{insert}_i : i \in [n]\}$. Note that $\texttt{insert}_n = \mathrm{Id}$. Specifically, we have $q(\sigma_t = \texttt{insert}_i) = 1/n$ when $i \neq n$ and $q(\sigma_t = \mathrm{Id}) = 1/n$.

Riffle Shuffles. Finally, we introduce the riffle shuffle, a method similar to how serious card players shuffle cards. The process begins by roughly cutting the deck into two halves and then interleaving the two halves together. A formal mathematical model of the riffle shuffle, known as the *GSR model*, was introduced by Gilbert and Shannon [11], and independently by Reeds [37]. The model is described as follows. A deck of n cards is cut into two piles according to binomial distribution, where the probability of having k cards in the top pile is $\binom{n}{k}/2^n$ for $0 \leq k \leq n$. The top pile is held in the left hand and the bottom pile in the right hand. The two piles are then riffled together such that, if there are A cards left in the left hand and B cards in the right hand, the probability that the next card drops from the left is $A/(A+B)$, and from right is $B/(A+B)$. We implement the riffle shuffles according to the GSR model. For simplicity, we will omit the term "GSR" when referring to riffle shuffles hereafter.

There exists an exact formula for the probability over S_n obtained through one-step riffle shuffle. Let $\sigma \in S_n$. A *rising sequence* of σ is a subsequence of σ constructed by finding a maximal subset of indices $i_1 < i_2 < \cdots < i_j$ such that permuted values are contiguously increasing, *i.e.*, $\sigma(i_2) - \sigma(i_1) = \sigma(i_3) - \sigma(i_2) = \cdots = \sigma(i_j) - \sigma(i_{j-1}) = 1$. For example, the permutation $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 2 & 5 & 3 \end{pmatrix}$ has 2 rising sequences, *i.e.*, 123 (red) and 45 (blue). Note that a permutation has 1 rising sequence if and only if it is the identity permutation. Denoting by $q_{\mathrm{RS}}(\sigma)$ the probability of obtaining σ through one-step riffle shuffle, it is shown in [4] that

$$q_{\mathrm{RS}}(\sigma) = \frac{1}{2^n}\binom{n+2-r}{n} = \begin{cases} (n+1)/2^n & \text{if } \sigma = \mathrm{Id} \\ 1/2^n & \text{if } \sigma \text{ has two rising sequences} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where r is the number of rising sequences of σ. The support $\mathcal{S}$ is thus the set of all permutations with at most two rising sequences. We let the forward process be $q(\sigma_t) = q_{\mathrm{RS}}(\sigma_t)$ for all t.

3.1.2 Mixing Times and Cut-off Phenomenon

All of the above shuffling methods have the uniform distribution as the stationary distribution. However, they have different mixing times (*i.e.*, the time until the Markov chain is close to its stationary distribution measured by some distance), and there exist quantitative results on their mixing times. Let $q \in \{q_{\mathrm{RT}}, q_{\mathrm{RI}}, q_{\mathrm{RS}}\}$, and for $t \in \mathbb{N}$, let $q^{(t)}$ be the marginal distribution of the Markov chain after t shuffles. We describe the mixing time in terms of the total variation (TV) distance between two probability distributions, *i.e.*, $D_{\mathrm{TV}}(q^{(t)}, u)$, where u is the uniform distribution.

For all three shuffling methods, there exists a *cut-off phenomenon*, where $D_{\mathrm{TV}}(q^{(t)}, u)$ stays around 1 for initial steps and then abruptly drops to values that are close to 0. The *cut-off time* is the time when the abrupt change happens. For the formal definition, we refer the readers to Definition 3.3 of

[38]. In [38], they also provided the cut-off time for random transposition, random insertion, and riffle shuffle, which are $\frac{n}{2}\log n$, $n\log n$, and $\frac{3}{2}\log_2 n$ respectively. Observe that the riffle shuffle reaches the cut-off much faster than the other two methods, which means it has a much faster mixing time. Therefore, we use the riffle shuffle in the forward process.

3.2 The Reverse Diffusion Process

We now model the *reverse process* as another Markov chain conditioned on the set of objects $\mathcal{X}$. We denote the set of realizable *reverse permutations* as $\mathcal{T}$, and the neighbours of X with respect to $\mathcal{T}$ as $N_{\mathcal{T}}(X) := \{Q_\sigma X : \sigma \in \mathcal{T}\}$. The conditional joint distribution is given by

$$p_\theta(X_{0:T}|\mathcal{X}) = p(X_T|\mathcal{X}) \prod_{t=1}^{T} p_\theta(X_{t-1}|X_t), \tag{4}$$

where $p_\theta(X_{t-1}|X_t) = \sum_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) p_\theta(\sigma'_t|X_t)$. To sample from $p(X_T|\mathcal{X})$, one simply samples a random permutation from the uniform distribution and then shuffle the objects accordingly to obtain X_T. $p(X_{t-1}|X_t, \sigma'_t)$ is again a delta distribution $\delta(X_{t-1} = Q_{\sigma'_t} X_t)$. We have

$$p_\theta(X_{t-1}|X_t) = \begin{cases} p_\theta\left(\sigma'_t|X_t\right) & \text{if } X_{t-1} \in N_{\mathcal{T}}(X_t) \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where $X_{t-1} \in N_{\mathcal{T}}(X_t)$ is equivalent to $\sigma'_t \in \mathcal{T}$ and $X_{t-1} = Q_{\sigma'_t} X_t$. In the following, we will introduce the specific design choices of the distribution $p_\theta(\sigma'_t|X_t)$.

3.2.1 Inverse Card Shuffling

A natural choice is to use the inverse operations of the aforementioned card shuffling operations in the forward process. Specifically, for the forward shuffling $\mathcal{S}$, we introduce their inverse operations $\mathcal{T} := \{\sigma^{-1} : \sigma \in \mathcal{S}\}$, from which we can parameterize $p_\theta(\sigma'_t|X_t)$.

Inverse Transposition. Since the inverse of a transposition is also a transposition, we can let $\mathcal{T} := \mathcal{S} = \{(i \quad j) : i \neq j \in [n]\} \cup \{\mathrm{Id}\}$. We define a distribution of inverse transposition (IT) over $\mathcal{T}$ using $n+1$ real-valued parameters $\mathbf{s} = (s_1, \ldots, s_n)$ and τ such that

$$p_{\mathrm{IT}}(\sigma) = \begin{cases} 1 - \phi(\tau) & \text{if } \sigma = \mathrm{Id} \\ \phi(\tau)\big(\psi(\mathbf{s}, \pi_{ij})_1 \psi(\mathbf{s}, \pi_{ij})_2 + \psi(\mathbf{s}, \pi_{ji})_1 \psi(\mathbf{s}, \pi_{ji})_2\big) & \text{if } \sigma = \begin{pmatrix} i & j \end{pmatrix} \text{ where } i \neq j, \end{cases} \tag{6}$$

where $\psi(\mathbf{s}, \pi)_i = \exp\left(s_{\pi(i)}\right) / \left(\sum_{k=i}^{n} \exp\left(s_{\pi(k)}\right)\right)$ and $\phi(\cdot)$ is the sigmoid function. π_{ij} is any permutation starting with i and j, *i.e.*, $\pi_{ij}(1) = i$ and $\pi_{ij}(2) = j$. π_{ji} is any permutation starting with j and i, *i.e.*, $\pi_{ji}(1) = j$ and $\pi_{ji}(2) = i$.

Inverse Insertion. For the random insertion, the inverse operation is to insert some piece to the end. Let $\texttt{inverse\_insert}_i$ denote the permutation that moves the i^{th} component to the end, and let $\mathcal{T} := \{\texttt{inverse\_insert}_i : i \in [n]\}$. We define a categorial distribution of inverse insertion (II) over $\mathcal{T}$ using parameters $\mathbf{s} = (s_1, \ldots, s_n)$ such that,

$$p_{\mathrm{II}}(\sigma = \texttt{inverse\_insert}_i) = \exp(s_i) / \Big(\textstyle\sum_{j=1}^{n} \exp(s_j)\Big). \tag{7}$$

Inverse Riffle Shuffle. In the riffle shuffle, the deck of card is first cut into two piles, and the two piles are riffled together. So to undo a riffle shuffle, we need to figure out which pile each card belongs to, *i.e.*, making a sequence of n binary decisions. We define the Inverse Riffle Shuffle (IRS) distribution using parameters $\mathbf{s} = (s_1, \ldots, s_n)$ as follows. Starting from the last (the n^{th}) object, each object i has probability $\phi(s_i)$ of being put on the top of the left pile. Otherwise, it falls on the top of the right pile. Finally, put the left pile on top of the right pile, which gives the shuffled result.

3.2.2 The Plackett-Luce Distribution and Its Generalization

Other than specific inverse shuffling methods to parameterize the reverse process, we also consider general distributions $p_\theta(\sigma'_t|X_t)$ whose support are the whole S_n, *i.e.*, $\mathcal{T} = S_n$.

The PL Distribution. A popular distribution over S_n is the Plackett-Luce (PL) distribution [35, 27], which is constructed from n real-valued scores $\mathbf{s} = (s_1, \ldots, s_n)$ as follows,

$$p_{\mathrm{PL}}(\sigma) = \prod_{i=1}^{n} \exp\left(s_{\sigma(i)}\right) / \left(\sum_{j=i}^{n} \exp\left(s_{\sigma(j)}\right)\right), \tag{8}$$

for all $\sigma \in S_n$. Intuitively, $(s_1, \ldots, s_n)$ represents the preference given to each index in $[n]$. To sample from $\mathrm{PL}_{\mathbf{s}}$, we first sample $\sigma(1)$ from $\mathrm{Cat}(n, \mathrm{softmax}(\mathbf{s}))$. Then we remove $\sigma(1)$ from the list and sample $\sigma(2)$ from the categorical distribution corresponding to the rest of the scores (logits). We continue in this manner until we have sampled $\sigma(1), \ldots, \sigma(n)$. By [7], the mode of the PL distribution is the permutation that sorts $\mathbf{s}$ in descending order.

The Generalized PL (GPL) Distribution. We also propose a generalization of the PL distribution, referred to as *Generalized Plackett-Luce (GPL) Distribution*. Unlike the PL distribution, which uses a set of n scores, the GPL distribution uses n^2 scores $\{\mathbf{s}_1, \cdots, \mathbf{s}_n\}$, where each $\mathbf{s}_i = \{s_{i,1}, \ldots, s_{i,n}\}$ consists of n scores. The GPL distribution is constructed as follows,

$$p_{\mathrm{GPL}}(\sigma) := \prod_{i=1}^{n} \exp\left(s_{i,\sigma(i)}\right) / \left(\sum_{j=i}^{n} \exp\left(s_{i,\sigma(j)}\right)\right). \tag{9}$$

Sampling of the GPL distribution begins with sampling $\sigma(1)$ using n scores $\mathbf{s}_1$. For $2 \le i \le n$, we remove $i-1$ scores from $\mathbf{s}_i$ that correspond to $\sigma(1), \ldots, \sigma(i-1)$ and sample $\sigma(i)$ from a categorical distribution constructed from the remaining $n-i+1$ scores in $\mathbf{s}_i$. It is important to note that the family of PL distributions is a strict subset of the GPL family. Since the GPL distribution has more parameters than the PL distribution, it is expected to be more expressive. In fact, when considering their ability to express the delta distribution, which is the target distribution for many permutation learning problems, we have the following result.

Proposition 1. *The PL distribution cannot exactly represent a delta distribution. That is, there does not exist an* $\mathbf{s}$ *such that* $p_{\mathrm{PL}} = \delta_\sigma$ *for any* $\sigma \in S_n$*, where* $\delta_\sigma(\sigma) = 1$ *and* $\delta_\sigma(\pi) = 0$ *for all* $\pi \neq \sigma$*. But the GPL distribution can represent a delta distribution exactly.*

3.3 Network Architecture and Training

We now briefly introduce how to use neural networks to parameterize the above distributions used in the reverse process. At any time t, given $X_t \in \mathbb{R}^{n \times d}$, we use a neural network with parameters θ to construct $p_\theta(\sigma_t' | X_t)$. In particular, we treat n rows of X_t as n tokens and use a Transformer architecture along with the time embedding of t and the positional encoding to predict the previously mentioned scores. For example, for the GPL distribution, to predict n^2 scores, we introduce n dummy tokens that correspond to the n permuted output positions. We then perform a few layers of masked self-attention ($2n \times 2n$) to obtain the token embedding $Z_1 \in \mathbb{R}^{n \times d_{\text{model}}}$ corresponding to n input tokens and $Z_2 \in \mathbb{R}^{n \times d_{\text{model}}}$ corresponding to n dummy tokens. Finally, the GPL score matrix is obtained as $S_\theta = Z_1 Z_2^\top \in \mathbb{R}^{n \times n}$. Since the aforementioned distributions have different numbers of scores, the specific architectures of the Transformer differ. We provide more details in Appendix B.

To learn the diffusion model, we maximize the following variational lower bound:

$$\mathbb{E}_{p_{\text{data}}(X_0, \mathcal{X})}\Big[\log p_\theta(X_0 | \mathcal{X})\Big] \ge \mathbb{E}_{p_{\text{data}}(X_0, \mathcal{X}) q(X_{1:T} | X_0, \mathcal{X})}\left[\log p(X_T | \mathcal{X}) + \sum_{t=1}^{T} \log \frac{p_\theta(X_{t-1} | X_t)}{q(X_t | X_{t-1})}\right]. \tag{10}$$

In practice, one can draw samples to obtain the Monte Carlo estimation of the lower bound. Due to the complexity of shuffling transition in the forward process, we can not obtain $q(X_t | X_0)$ analytically, as is done in common diffusion models [16, 3]. Therefore, we have to run the forward process to collect samples. Fortunately, it is efficient as the forward process only involves shuffling integers. We include more training details in Appendix E.

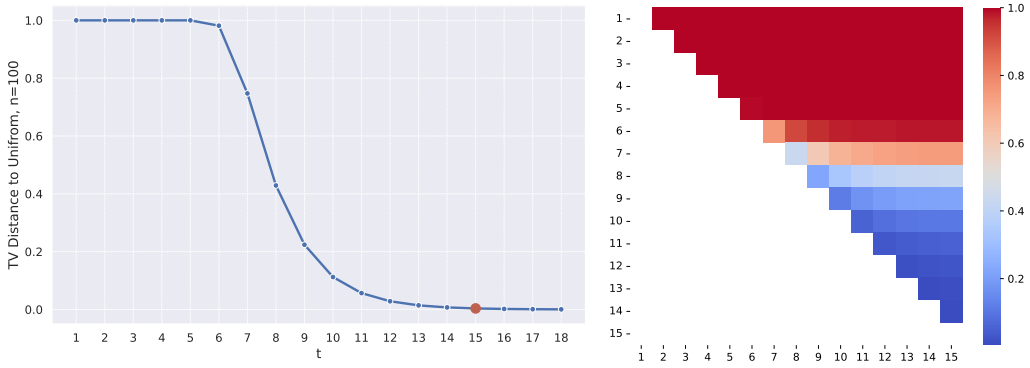
3.4 Denoising Schedule via Merging Reverse Steps

If one merges some steps in the reverse process, sampling and learning would be faster and more memory efficient. The variance of the training loss could also be reduced. Specifically, at time t of the reverse process, instead of predicting $p_\theta(X_{t-1} | X_t)$, we can predict $p_\theta(X_{t'} | X_t)$ for any $0 \le t' < t$. Given a sequence of timesteps $0 = t_0 < \cdots < t_k = T$, we can now model the reverse process as

$$p_\theta(X_{t_0}, \ldots, X_{t_k} | \mathcal{X}) = p(X_T | \mathcal{X}) \prod_{i=1}^{k} p_\theta(X_{t_{i-1}} | X_{t_i}). \tag{11}$$

To align with the literature of diffusion models, we call the list $[t_0, \ldots, t_k]$ the *denoising schedule*. After incorporating the denoising schedule in Eq. (10), we obtain the loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{p_{\text{data}}(X_0, \mathcal{X})} \mathbb{E}_{q(X_{1:T} | X_0, \mathcal{X})}\left[-\log p(X_T | \mathcal{X}) - \sum_{i=1}^{k} \log \frac{p_\theta(X_{t_{i-1}} | X_{t_i})}{q(X_{t_i} | X_{t_{i-1}})}\right]. \tag{12}$$



(a) (b)

Figure 2: (a) $D_{\mathrm{TV}}(q_{\mathrm{RS}}^{(t)}, u)$ computed using Eq. (14). We choose $T = 15$ (red dot) based on the threshold 0.005. (b) A heatmap for $D_{\mathrm{TV}}(q_{\mathrm{RS}}^{(t)}, q_{\mathrm{RS}}^{(t')})$ for $n = 100$ and $1 \leq t < t' \leq 15$, computed using Eq. (13). Rows are t and columns are t'. We choose the denoising schedule $[0, 8, 10, 15]$.

Note that although we may not have the analytical form of $q(X_{t_i}|X_{t_{i-1}})$, we can draw samples from it. Merging is feasible if the support of $p_\theta(X_{t_{i-1}}|X_{t_i})$ is equal or larger than the support of $q(X_{t_i}|X_{t_{i-1}})$; otherwise, the inverse of some forward permutations would be almost surely unrecoverable. Therefore, we can implement a non-trivial denoising schedule (*i.e.*, $k < T$), when $p_\theta(\sigma'_t|X_t)$ follows the PL or GPL distribution, as they have whole S_n as their support. However, merging is not possible for inverse shuffling methods, as their support is smaller than that of the corresponding multi-step forward shuffling. To design a successful denoising schedule, we first describe the intuitive principles and then provide some theoretical insights. 1) The length of forward diffusion T should be minimal so long as the forward process approaches the uniform distribution. 2) If distributions of X_t and X_{t+1} are similar, we should merge these two steps. Otherwise, we should not merge them, as it would make the learning problem harder.

To quantify the similarity between distributions shown in 1) and 2), the TV distance is commonly used in the literature. In particular, we can measure $D_{\mathrm{TV}}(q^{(t)}, q^{(t')})$ for $t \neq t'$ and $D_{\mathrm{TV}}(q^{(t)}, u)$, where $q^{(t)}$ is the distribution at time t in the forward process and u is the uniform distribution. For riffle shuffles, the total variation distance can be computed exactly. Specifically, we first introduce the *Eulerian Numbers* $A_{n,r}$ [32], *i.e.*, the number of permutations in S_n that have exactly r rising sequences where $1 \leq r \leq n$. $A_{n,r}$ can be computed using the following recursive formula $A_{n,r} = rA_{n-1,r} + (n - r + 1)A_{n-1,r-1}$ where $A_{1,1} = 1$. We then have the following result.

Proposition 2. *Let $t \neq t'$ be positive integers. Then*

$$D_{\mathrm{TV}}\left(q_{\mathrm{RS}}^{(t)}, q_{\mathrm{RS}}^{(t')}\right) = \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|\frac{1}{2^{tn}}\binom{n + 2^t - r}{n} - \frac{1}{2^{t'n}}\binom{n + 2^{t'} - r}{n}\right|, \tag{13}$$

and

$$D_{\mathrm{TV}}\left(q_{\mathrm{RS}}^{(t)}, u\right) = \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|\frac{1}{2^{tn}}\binom{n + 2^t - r}{n} - \frac{1}{n!}\right|. \tag{14}$$

Note that Eq. (14) was originally given in [19]. We restate it here for completeness. Once the Eulerian numbers are precomputed, the TV distances can be computed in $O(n)$ time instead of $O(n!)$. Through extensive experiments, we have the following empirical observation. For the principle 1), choosing T so that $D_{\mathrm{TV}}(q_{\mathrm{RS}}^{(T)}, u) \approx 0.005$ yields good results. For the principle 2), a denoising schedule $[t_0, \ldots, t_k]$ with $D_{\mathrm{TV}}(q_{\mathrm{RS}}^{(t_i)}, q_{\mathrm{RS}}^{(t_{i+1})}) \approx 0.3$ for most i works well. We show an example on sorting $n = 100$ four-digit MNIST images in Fig. 2.

3.5 Reverse Process Decoding

We now discuss how to decode predictions from the reverse process at test time. In practice, one is often interested in the most probable state or a few states with high probabilities under $p_\theta(X_0|\mathcal{X})$. However, since we can only draw samples from $p_\theta(X_0|\mathcal{X})$ via running the reverse process, exact decoding is intractable. The simplest approximated method is greedy search, *i.e.*, successively finding the mode or an approximated mode of $p_\theta(X_{t_{i-1}}|X_{t_i})$. Another approach is beam search, which

Method	Metrics	Noisy MNIST					CIFAR-10		
		2×2	3×3	4×4	5×5	6×6	2×2	3×3	4×4
Gumbel-Sinkhorn Network [29]	Kendall-Tau ↑	0.9984	0.6908	0.3578	0.2430	0.1755	0.8378	0.5044	**0.4016**
	Accuracy (%)	99.81	44.65	00.86	0.00	0.00	76.54	6.07	0.21
	Correct (%)	99.91	80.20	49.51	26.94	14.91	86.10	43.59	25.31
	RMSE ↓	**0.0022**	0.1704	0.4572	0.8915	1.0570	0.3749	0.9590	1.0960
	MAE ↓	0.0003	0.0233	0.1005	0.3239	0.4515	0.1368	0.5320	0.6873
DiffSort [34]	Kendall-Tau ↑	0.9931	0.3054	0.0374	0.0176	0.0095	0.6463	0.1460	0.0490
	Accuracy (%)	99.02	5.56	0.00	0.00	0.00	59.18	0.96	0.00
	Correct (%)	99.50	42.25	10.77	6.39	3.77	75.48	27.87	12.27
	RMSE ↓	0.0689	1.0746	1.3290	1.4883	1.5478	0.7389	1.2691	1.3876
	MAE ↓	0.0030	0.4283	0.6531	0.8204	0.8899	0.2800	0.8123	0.9737
Error-free DiffSort [20]	Kendall-Tau ↑	0.9899	0.2014	0.0100	0.0034	-0.0021	0.6604	0.1362	0.0318
	Accuracy (%)	98.62	0.82	0.00	0.00	0.00	60.96	0.68	0.00
	Correct (%)	99.28	32.65	7.40	4.39	2.50	75.99	26.75	10.33
	RMSE ↓	0.0814	1.1764	1.3579	1.5084	1.5606	0.7295	1.2820	1.4095
	MAE ↓	0.0041	0.5124	0.6818	0.8424	0.9041	0.2731	0.8260	0.9990
Symmetric Diffusers (Ours)	Kendall-Tau ↑	**0.9992**	**0.8126**	**0.4859**	**0.2853**	**0.1208**	**0.9023**	**0.8363**	0.2518
	Accuracy (%)	**99.88**	**57.38**	**1.38**	0.00	0.00	**90.15**	**70.94**	**0.64**
	Correct (%)	**99.94**	**86.16**	**58.51**	**37.91**	**18.54**	**92.99**	**86.84**	**34.69**
	RMSE ↓	0.0026	**0.0241**	**0.1002**	**0.2926**	**0.4350**	**0.3248**	**0.3892**	**0.8953**
	MAE ↓	**0.0001**	**0.0022**	**0.0130**	**0.0749**	**0.1587**	**0.0651**	**0.0977**	**0.5044**

Table 1: Results (averaged over 5 runs) on solving the jigsaw puzzle on Noisy MNIST and CIFAR10.

Method	Metrics	Sequence Length							
		3	5	7	9	15	32	52	100
DiffSort [34]	Kendall-Tau ↑	0.930	0.898	0.864	0.801	0.638	0.535	0.341	0.166
	Accuracy (%)	93.8	83.9	71.5	52.2	10.3	0.2	0.0	0.0
	Correct (%)	95.8	92.9	90.1	85.2	82.3	61.8	42.8	23.2
Error-free DiffSort [20]	Kendall-Tau ↑	0.974	**0.967**	**0.962**	**0.952**	**0.938**	**0.879**	0.170	0.140
	Accuracy (%)	97.7	95.3	92.9	89.6	**83.1**	**57.1**	0.0	0.0
	Correct (%)	98.4	**97.7**	**97.2**	**96.3**	**95.1**	**90.1**	24.2	20.1
Symmetric Diffusers (Ours)	Kendall-Tau ↑	**0.976**	**0.967**	0.959	0.950	0.932	0.858	**0.786**	**0.641**
	Accuracy (%)	**98.0**	**95.5**	**92.9**	**90.0**	82.6	55.1	**27.4**	**4.5**
	Correct (%)	**98.5**	97.6	96.8	96.1	94.5	88.3	**82.1**	**69.3**

Table 2: Results (averaged over 5 runs) on the four-digit MNIST sorting benchmark.

maintains a dynamic buffer of k candidates with highest probabilities. Nevertheless, for one-step reverse transitions like the GPL distribution, even finding the mode is intractable. To address this, we employ a hierarchical beam search that performs an inner beam search within n^2 scores at each step of the outer beam search. Further details are provided in Appendix C.

4 Experiments

We now demonstrate the general applicability and effectiveness of our model through a variety of experiments, including sorting 4-digit MNIST numbers, solving jigsaw puzzles, and addressing traveling salesman problems. Additional details are provided in the appendix due to space constraints.

4.1 Sorting 4-digit MNIST Images

We first evaluate our SymmetricDiffusers on the four-digit MNIST sorting benchmark, a well-established testbed for differentiable sorting [5, 8, 13, 20, 33, 34]. Each four-digit image in this benchmark is obtained by concatenating 4 individual images from MNIST. For evaluation, we employ several metrics to compare methods, including Kendall-Tau coefficient (measuring the correlation between rankings), accuracy (percentage of images perfectly reassembled), and correctness (percentage of pieces that are correctly placed).

Ablation Study. We conduct an ablation study to verify our design choices for reverse transition and decoding strategies. As shown in Table 3, combining PL with either beam search (BS) or greedy search yields good results in terms of Kendall-Tau and correctness metrics. In contrast, the IRS (inverse riffle shuffle) method, along with greedy search, performs poorly across all metrics, showing the limitations of IRS in handling complicated sorting tasks. Finally, combining GPL and BS achieves the best accuracy in correctly sorting the entire sequence of images. Given that accuracy is the most

	GPL + BS	GPL + Greedy	PL + Greedy	PL + BS	IRS + Greedy
Kendall-Tau ↑	0.786	**0.799**	**0.799**	0.797	0.390
Accuracy (%)	**27.4**	24.4	26.4	26.4	0.6
Correct (%)	82.1	81.6	**83.3**	83.1	44.6

Table 3: Ablation study on transitions of reverse diffusion and decoding strategies. Results are averaged over three runs on sorting 52 four-digit MNIST images.

Method	**OR Solvers**				**Learning-Based Models**		
	Gurobi [14]	Concorde [1]	LKH-3 [15]	2-Opt [25]	GCN* [18]	DIFUSCO* [43]	Ours
Tour Length ↓	**3.842**	3.843	**3.842**	4.020	3.850	3.883	**3.849**
Optimality Gap (%) ↓	0.00	0.00	0.00	4.64	0.21	1.07	0.18

Table 4: Results on TSP-20. * means we remove the post-processing heuristics for a fair comparison.

challenging metric to improve, we selecte GPL and BS for all remaining experiments. More ablation study (*e.g.*, denoising schedule) is provided in Appendix E.2.

Full Results. From Table 2, we can see that Error-free DiffSort achieves the best performance in sorting sequences with lengths up to 32. However, its performances drop significantly with long sequences (*e.g.*, length of 52 or 100). Meanwhile, DiffSort performs the worse due to the error accumulation of its soft differentiable swap function [20, 33]. In contrast, our method is on par with Error-free DiffSort in sorting short sequences and significantly outperforms others on long sequences.

4.2 Jigsaw Puzzle

We then explore image reassembly from segmented "jigsaw" puzzles [29, 31, 39]. We evaluate the performance using the MNIST and the CIFAR10 datasets, which comprises puzzles of up to 6×6 and 4×4 pieces respectively. We add slight noise to pieces from the MNIST dataset to ensure background pieces are distinctive. To evaluate our models, we use Kendall-Tau coefficient, accuracy, correctness, RMSE (root mean square error of reassembled images), and MAE (mean absolute error) as metrics.

Table 1 presents results comparing our method with the Gumbel-Sinkhorn Network[29], Diffsort [34], and Error-free Diffsort [20]. DiffSort and Error-free DiffSort are primarily designed for sorting high-dimensional ordinal data which have clearly different patterns. Since jigsaw puzzles on MNIST and CIFAR10 contain pieces that are visually similar, these methods do not perform well. The Gumbel-Sinkhorn performs better for tasks involving fewer than 4×4 pieces. In more challenging scenarios (*e.g.*, 5×5 and 6×6), our method significantly outperforms all competitors.

4.3 The Travelling Salesman Problem

At last, we explore the travelling salesman problem (TSP) to demonstrate the general applicability of our model. TSPs are classical NP-complete combinatorial optimization problems which are solved using integer programming or heuristic solvers [2, 12]. There exists a vast literature on learning-based models to solve TSPs [22, 23, 18, 17, 6, 24, 10, 36, 21, 43, 30]. They often focus on the Euclidean TSPs, which are formulated as follows. Let $V = \{v_1, \ldots, v_n\}$ be points in $\mathbb{R}^2$. We need to find some $\sigma \in S_n$ such that $\sum_{i=1}^{n} \|v_{\sigma(i)} - v_{\sigma(i+1)}\|_2$ is minimized, where we let $\sigma(n+1) := \sigma(1)$. Further experimental details are provided in Appendix B.

We compare with operations research (OR) solvers and other learning based approaches on TSP instances with 20 nodes. The metrics are the total tour length and the optimality gap. Given the ground truth (GT) length produced by the best OR solver, the optimality gap is given by $\big($predicted length $-$ (GT length)$\big)/$(GT length). As shown in Table 4, SymmetricDiffusers achieves comparable results with both OR solvers and the state-of-the-art learning-based methods.

5 Conclusion

In this paper, we introduce a novel discrete diffusion model over finite symmetric groups. We identify the riffle shuffle as an effective forward transition and provide empirical rules for selecting the diffusion length. Additionally, we propose a generalized PL distribution for the reverse transition, which is provably more expressive than the PL distribution. We further introduce a theoretically grounded "denoising schedule" to improve sampling and learning efficiency. Extensive experiments verify the effectiveness of our proposed model. In the future, we are interested in generalizing our model to general finite groups and exploring diffusion models on Lie groups.

References

[1] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Concorde tsp solver, 2006.

[2] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, Sep 1998.

[3] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces, 2023.

[4] Dave Bayer and Persi Diaconis. Trailing the dovetail shuffle to its lair. *The Annals of Applied Probability*, 2(2):294 – 313, 1992.

[5] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pages 950–959. PMLR, 2020.

[6] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem, 2021.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA, 2007. Association for Computing Machinery.

[8] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems*, 32, 2019.

[9] Persi Diaconis. Group representations in probability and statistics. *Lecture notes-monograph series*, 11:i–192, 1988.

[10] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances, 2021.

[11] E. N. Gilbert. Theory of shuffling. *Bell Telephone Laboratories Memorandum*, 1955.

[12] Teofilo F. Gonzalez. *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2007.

[13] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, 2018.

[14] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[15] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems, Dec 2017.

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[17] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning tsp requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming*, 2021.

[18] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem, 2019.

[19] Shihan Kanungo. Mixing time estimates for the riffle shuffle. *Euler Circle*, 2020.

[20] Jungtaek Kim, Jeongbeen Yoon, and Minsu Cho. Generalized neural sorting networks with error-free differentiable swap functions. In *International Conference on Learning Representations (ICLR)*, 2024.

[21] Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization, 2023.

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

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

[24] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning, 2021.

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

[26] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[27] R. D. Luce. *Individual Choice Behavior*. John Wiley, 1959.

[28] Colin L Mallows. Non-null ranking models. i. *Biometrika*, 44(1/2):114–130, 1957.

[29] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018.

[30] Yimeng Min, Yiwei Bai, and Carla P. Gomes. Unsupervised learning for solving the travelling salesman problem, 2024.

[31] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer, 2016.

[32] OEIS Foundation Inc. The eulerian numbers, entry a008292 in the On-Line Encyclopedia of Integer Sequences, 2024. Published electronically at `http://oeis.org/A008292`.

[33] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In *International conference on machine learning*, pages 8546–8555. PMLR, 2021.

[34] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Monotonic differentiable sorting networks. In *International Conference on Learning Representations (ICLR)*, 2022.

[35] R. L. Plackett. The analysis of permutations. *Applied Statistics*, 24(2):193 – 202, 1975.

[36] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems, 2022.

[37] J. Reeds. Theory of shuffling. *Unpublished Manuscript*, 1981.

[38] Laurent Saloff-Coste. *Random Walks on Finite Groups*, pages 263–346. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[39] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3949–3957, 2017.

[40] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

[41] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

[42] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[43] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization, 2023.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[45] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation, 2023.

[46] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

A Additional Details of the GSR Riffle Shuffle Model

There are many equivalent definitions of the GSR riffle shuffle. Here we also introduce the *Geometric Description* [4], which is easy to implement (and is how we implement riffle shuffles in our experiments). We first sample n points in the unit interval $[0, 1]$ uniformly and independently, and suppose the points are labeled in order as $x_1 < x_2 < \cdots < x_n$. Then, the permutation that sorts the points $\{2x_1\}, \ldots, \{2x_n\}$ follows the GSR distribution, where $\{x\} := x - \lfloor x \rfloor$ is the fractional part of x.

B Details of Our Network Architecture

We now discuss how to use neural networks to produce the parameters of the distributions discussed in Section 3.2.1 and 3.2.2. Fix time t, and suppose $X_t = \left(\mathbf{x}_1^{(t)}, \ldots, \mathbf{x}_n^{(t)}\right)^\top \in \mathbb{R}^{n \times d}$. Let $\texttt{encoder}_\theta$ be an object-specific encoder such that $\texttt{encoder}_\theta(X_t) \in \mathbb{R}^{n \times d_{\text{model}}}$. For example, $\texttt{encoder}_\theta$ can be a CNN if X_t is an image. Let

$$Y_t := \texttt{encoder}_\theta(X_t) + \texttt{time\_embd}(t) = \left(\mathbf{y}_1^{(t)}, \ldots, \mathbf{y}_n^{(t)}\right)^\top \in \mathbb{R}^{n \times d_{\text{model}}}, \tag{15}$$

where time_embd is the sinusoidal time embedding. Then, we would like to feed the embeddings into a Transformer encoder [44]. Let $\texttt{transformer\_encoder}_\theta$ be the encoder part of the Transformer architecture. However, each of the distributions we discussed previously has different number of parameters, so we will have to discuss them separately.

Inverse Transposition. For Inverse Transposition, we have $n + 1$ parameters. To obtain $n + 1$ tokens from $\texttt{transformer\_encoder}_\theta$, we append a dummy token of 0's to Y_t. Then we input $\left(\mathbf{y}_1^{(t)}, \ldots, \mathbf{y}_n^{(t)}, 0\right)^\top$ into $\texttt{transformer\_encoder}_\theta$ to obtain $Z \in \mathbb{R}^{(n+1) \times d_{\text{model}}}$. Finally, we apply an MLP to obtain $(s_1, \ldots, s_n, k) \in \mathbb{R}^{n+1}$.

Inverse Insertion, Inverse Riffle Shuffle, PL Distribution. These three distributions all require exactly n parameters, so we can directly feed Y_t into $\texttt{transformer\_encoder}_\theta$. Let the output of $\texttt{transformer\_encoder}_\theta$ be $Z \in \mathbb{R}^{n \times d_{\text{model}}}$, where we then apply an MLP to obtain the scores $\mathbf{s}_\theta \in \mathbb{R}^n$.

The GPL Distribution. The GPL distribution requires n^2 parameters. We first append n dummy tokens of 0's to Y_t, with the intent that the j^{th} dummy token would learn information about the j^{th} column of the GPL parameter matrix, which represents where the j^{th} component should be placed. We then pass $\left(\mathbf{y}_1^{(t)}, \ldots, \mathbf{y}_n^{(t)}, 0, \ldots, 0\right)^\top \in \mathbb{R}^{2n \times d_{\text{model}}}$ to $\texttt{transformer\_encoder}_\theta$. When computing attention, we further apply a $2n \times 2n$ attention mask

$$M := \begin{bmatrix} 0 & A \\ 0 & B \end{bmatrix}, \text{ where } A \text{ is an } n \times n \text{ matrix of } -\infty,\ B = \begin{bmatrix} -\infty & -\infty & \cdots & -\infty \\ 0 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\infty \end{bmatrix} \text{ is } n \times n.$$

The reason for having B as an upper triangular matrix of $-\infty$ is that information about the j^{th} component should only require information from the previous components. Let

$$\texttt{transformer\_encoder}_\theta(Y_t, M) = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix},$$

where $Z_1, Z_2 \in \mathbb{R}^{n \times d_{\text{model}}}$. Finally, we obtain the GPL parameter matrix as $S_\theta = Z_1 Z_2^\top \in \mathbb{R}^{n \times n}$.

For hyperparameters, we refer the readers to Appendix E.4.

C Additional Details of Decoding

Greedy Search. At each timestep t_i in the denoising schedule, we can greedily obtain or approximate the mode of $p_\theta(X_{t_{i-1}} | X_{t_i})$. We can then use the (approximated) mode $X_{t_{i-1}}$ for the next timestep $p_\theta(X_{t_{i-2}} | X_{t_{i-1}})$. Note that the final X_0 obtained using such a greedy heuristic may not necessarily be the mode of $p_\theta(X_0 | \mathcal{X})$.

Beam Search. We can use beam search to improve the greedy approach. The basic idea is that, at each timestep t_i in the denoising schedule, we compute or approximate the top-k-most-probable results from $p_\theta(X_{t_{i-1}}|X_{t_i})$. For each of the top-$k$ results, we sample top-k from $p_\theta(X_{t_{i-2}}|X_{t_{i-1}})$. Now we have k^2 candidates for $X_{t_{i-2}}$, and we only keep the top k of the k^2 candidates.

However, it is not easy to obtain the top-k-most-probable results for some of the distributions. Here we provide an algorithm to approximate top-k of the PL and the GPL distribution. Since the PL distribution is a strict subset of the GPL distribution, it suffices to only consider the GPL distribution with parameter matrix S. The algorithm for approximating top-k of the GPL distribution is another beam search. We first pick the k largest elements from the first row of S. For each of the k largest elements, we pick k largest elements from the second row of S, excluding the corresponding element picked in the first row. We now have k^2 candidates for the first two elements of a permutation, and we only keep the top-k-most-probable candidates. We then continue in this manner.

D Proofs

Proposition 1. *The PL distribution cannot exactly represent a delta distribution. That is, there does not exist an* $\mathbf{s}$ *such that* $p_{\mathrm{PL}} = \delta_\sigma$ *for any* $\sigma \in S_n$*, where* $\delta_\sigma(\sigma) = 1$ *and* $\delta_\sigma(\pi) = 0$ *for all* $\pi \neq \sigma$*. But the GPL distribution can represent a delta distribution exactly.*

Proof. Assume for a contradiction that there exists some $\sigma \in S_n$ and $\mathbf{s}$ such that $\mathrm{PL}_{\mathbf{s}} = \delta_\sigma$. Then we have

$$\prod_{i=1}^{n} \frac{\exp\left(s_{\sigma(i)}\right)}{\sum_{j=i}^{n} \exp\left(s_{\sigma(j)}\right)} = 1.$$

Since each of the term in the product is less than or equal to 1, we must have

$$\frac{\exp\left(s_{\sigma(i)}\right)}{\sum_{j=i}^{n} \exp\left(s_{\sigma(j)}\right)} = 1 \tag{16}$$

for all $i \in [n]$. In particular, we have

$$\frac{\exp\left(s_{\sigma(1)}\right)}{\sum_{j=1}^{n} \exp\left(s_{\sigma(j)}\right)} = 1,$$

which happens if and only if $s_{\sigma(j)} = -\infty$ for all $j \geq 2$. But this contradicts (16).

We then show that the GPL distribution can represent a delta distribution exactly. To see this, we fix $\sigma \in S_n$. For all $i \in [n]$, we let $s_{i,\sigma(i)} = 0$ and $s_{i,j} = -\infty$ for all $j \neq \sigma(i)$. Then $\mathrm{GPL}_{(s_{ij})} = \delta_\sigma$. □

Proposition 2. *Let* $t \neq t'$ *be positive integers. Then*

$$D_{\mathrm{TV}}\left(q_{\mathrm{RS}}^{(t)}, q_{\mathrm{RS}}^{(t')}\right) = \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|\frac{1}{2^{tn}}\binom{n+2^t-r}{n} - \frac{1}{2^{t'n}}\binom{n+2^{t'}-r}{n}\right|, \tag{13}$$

and

$$D_{\mathrm{TV}}\left(q_{\mathrm{RS}}^{(t)}, u\right) = \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|\frac{1}{2^{tn}}\binom{n+2^t-r}{n} - \frac{1}{n!}\right|. \tag{14}$$

Proof. Let $\sigma \in S_n$. It was shown in [4] that

$$q_{\mathrm{RS}}^{(t)}(\sigma) = \frac{1}{2^{tn}} \cdot \binom{n+2^t-r}{n},$$

where r is the number of rising sequences of σ. Note that if two permutations have the same number of rising sequences, then they have equal probability. Hence, we have

$$\begin{aligned} D_{\mathrm{TV}}\left(q_{\mathrm{RS}}^{(t)} - q_{\mathrm{RS}}^{(t')}\right) &= \frac{1}{2}\sum_{\sigma\in S_n}\left|q_{\mathrm{RS}}^{(t)}(\sigma) - q_{\mathrm{RS}}^{(t')}(\sigma)\right| = \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|q_{\mathrm{RS}}^{(t)}(\sigma) - q_{\mathrm{RS}}^{(t')}(\sigma)\right| \\ &= \frac{1}{2}\sum_{r=1}^{n} A_{n,r}\left|\frac{1}{2^{tn}}\binom{n+2^t-r}{n} - \frac{1}{2^{t'n}}\binom{n+2^{t'}-r}{n}\right|, \end{aligned}$$

as claimed. For (14), replace $q_{\mathrm{RS}}^{(t')}(\sigma)$ with $u(\sigma) = \frac{1}{n!}$ in the above derivations. □

E Additional Details on Experiments

E.1 Datasets

Jigsaw Puzzle. We created the Noisy MNIST dataset by adding *i.i.d.* Gaussian noise with a mean of 0 and a standard deviation of 0.01 to each pixel of the MNIST images. No noise was added to the CIFAR-10 images. The noisy images are then saved as the Noisy MNIST dataset. During training, each image is divided into $n \times n$ patches. A permutation is then sampled uniformly at random to shuffle these patches. The training set for Noisy MNIST comprises 60,000 images, while the CIFAR-10 training set contains 10,000 images. The Noisy MNIST test set, which is pre-shuffled, also includes 10,000 images. The CIFAR-10 test set, which shuffles images on the fly, contains 10,000 images as well.

Sort 4-Digit MNIST Numbers. For each training epoch, we generate 60,000 sequences of 4-digit MNIST images, each of length n, constructed dynamically on the fly. These 4-digit MNIST numbers are created by concatenating four MNIST images, each selected uniformly at random from the entire MNIST dataset, which consists of 60,000 images. For testing purposes, we similarly generate 10,000 sequences of n 4-digit MNIST numbers on the fly.

TSP. We take the TSP-20 dataset from [17] [1]. The train set consists of 1,512,000 graphs with 20 nodes, where each node is an *i.i.d.* sample from the unit square $[0, 1]^2$. The labels are optimal TSP tours provided by the Concorde solver [1]. The test set consists of 1,280 graphs with 20 nodes, with ground truth tour generated by the Concorde solver as well.

E.2 Ablation Studies

Choices for Reverse Transition and Decoding Strategies. As demonstrated in Table 5, we have explored various combinations of forward and inverse shuffling methods across tasks involving different sequence lengths. Both GPL and PL consistently excel in all experimental scenarios, highlighting their robustness and effectiveness. It is important to note that strategies such as random transposition and random insertion paired with their respective inverse operations, are less suitable for tasks with longer sequences. This limitation is attributed to the prolonged mixing times required by these two shuffling methods, a challenge that is thoroughly discussed in Section 3.1.2.

Denoising Schedule. We also conduct an ablation study on how we should merge reverse steps. As shown in Table 6, the choice of the denoising schedule can significantly affect the final performance. In particular, for $n = 100$ on the Sort 4-Digit MNIST Numbers task, the fact that $[0, 15]$ has 0 accuracy justifies our motivation to use diffusion to break down learning into smaller steps. The result we get also matches with our proposed heuristic in Section 3.4.

E.3 Latent Loss in Jigsaw Puzzle

In the original setup of the Jigsaw Puzzle experiment using the Gumbel-Sinkhorn network [29], the permutations are latent. That is, the loss function in Gumbel-Sinkhorn is a pixel-level MSE loss and does not use the ground truth permutation label. However, our loss function (12) actually (implicitly) uses the ground truth permutation that maps the shuffled image patches to their original order. Therefore, for fair comparison with the Gumbel-Sinkhorn network in the Jigsaw Puzzle experiment, we modify our loss function so that it does not use the ground truth permutation. Recall from Section 3.2 that we defined

$$p_\theta(X_{t-1}|X_t) = \sum_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) p_\theta(\sigma'_t|X_t). \tag{17}$$

In our original setup, we defined $p(X_{t-1}|X_t, \sigma'_t)$ as a delta distribution $\delta(X_{t-1} = Q_{\sigma'_t} X_t)$, but this would require that we know the permutation that turns X_{t-1} to X_t, which is part of the ground truth. So instead, we parameterize $p(X_{t-1}|X_t, \sigma'_t)$ as a Gaussian distribution $\mathcal{N}\big(X_{t-1}|Q_{\sigma_t} X_t, I\big)$. At the same time, we note that to find the gradient of (12), it suffices to find the gradient of the log of (17).

[1] `https://github.com/chaitjo/learning-tsp?tab=readme-ov-file`

		Sequence Length		
		9	32	52
RS (forward) + GPL (reverse) + greedy	Denoising Schedule	$[0, 3, 5, 9]$	$[0, 5, 7, 12]$	$[0, 5, 6, 7, 10, 13]$
	Kendall-Tau ↑	0.948	0.857	0.779
	Accuracy (%)	89.4	54.8	24.4
	Correct (%)	95.9	88.1	81.6
RS (forward) + PL (reverse) + greedy	Denoising Schedule	$[0, 3, 5, 9]$	$[0, 5, 7, 12]$	$[0, 5, 6, 7, 10, 13]$
	Kendall-Tau	0.953	0.867	0.799
	Accuracy (%)	90.9	56.4	26.4
	Correct (%)	96.4	89.0	83.3
RS (forward) + PL (reverse) + beam search	Denoising Schedule	$[0, 3, 5, 9]$	$[0, 5, 7, 12]$	$[0, 5, 6, 7, 10, 13]$
	Kendall-Tau ↑	0.955	0.869	0.797
	Accuracy (%)	91.1	57.2	26.4
	Correct (%)	96.5	89.2	83.1
RS (forward) + IRS (reverse) + greedy	T	9	12	13
	Kendall-Tau ↑	0.947	0.794	0.390
	Accuracy (%)	88.6	24.4	0.6
	Correct (%)	95.9	82.5	44.6
RT (forward) + IT (reverse) + greedy	T (using approx. $\frac{n}{2}\log n$)	15	55	105
	Kendall-Tau ↑	0.490		
	Accuracy (%)	18.0	Out of Memory	
	Correct (%)	59.5		
RI (forward) + II (reverse) + greedy	T (using approx. $n \log n$)	25	110	205
	Kendall-Tau ↑	0.954		
	Accuracy (%)	91.1	Out of Memory	
	Correct (%)	96.4		

Table 5: More results on sorting the 4-digit MNIST dataset using different combinations of forward process methods and reverse process methods. Results averaged over 3 runs with different seeds. RS: riffle shuffle; GPL: generalized Plackett-Luce; IRS: inverse riffle shuffle; RT: random transposition; IT: inverse transposition; RI: random insertion; II: inverse insertion.

Denoising Schedule	$[0, 15]$	$[0, 8, 9, 15]$	$[0, 7, 8, 9, 15]$	$[0, 7, 8, 10, 15]$	$[0, 8, 10, 15]$
Kendall-Tau ↑	0.000	0.316	0.000	0.000	**0.646**
Accuracy (%)	0.0	0.0	0.0	0.0	**4.5**
Correct (%)	1.0	39.6	1.0	1.0	**69.8**

Table 6: Results of sorting 100 4-digit MNIST images using various denoising schedules with the combination of RS, GPL and beam search consistently applied.

We use the REINFORCE trick [46] to find the gradient of $\log p_\theta(X_{t-1}|X_t)$, which gives us

$$
\begin{aligned}
&\nabla_\theta \log p_\theta(X_{t-1}|X_t) \\
&= \frac{1}{\sum\limits_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) p_\theta(\sigma'_t|X_t)} \cdot \sum_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) \nabla_\theta p_\theta(\sigma'_t|X_t) \\
&= \frac{1}{\sum\limits_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) p_\theta(\sigma'_t|X_t)} \cdot \sum_{\sigma'_t \in \mathcal{T}} p(X_{t-1}|X_t, \sigma'_t) p_\theta(\sigma'_t|X_t) \big(\nabla_\theta \log p_\theta(\sigma_t|X_t)\big) \\
&= \frac{\mathbb{E}_{p_\theta(\sigma_t|X_t)}\Big[p(X_{t-1}|X_t, \sigma'_t) \nabla_\theta \log p_\theta(\sigma_t|X_t)\Big]}{\mathbb{E}_{p_\theta(\sigma_t|X_t)}\Big[p(X_{t-1}|X_t, \sigma'_t)\Big]} \\
&\approx \sum_{n=1}^{N} \frac{p\left(X_{t-1}|X_t, \sigma_t^{(n)}\right)}{\sum_{m=1}^{N} p\left(X_{t-1}|X_t, \sigma_t^{(m)}\right)} \cdot \nabla_\theta \log p_\theta\left(\sigma_t^{(n)}|X_t\right),
\end{aligned}
$$

where we have used Monte-Carlo estimation in the last step, and $\sigma_t^{(1)}, \ldots, \sigma_t^{(N)} \sim p_\theta(\sigma_t|X_t)$. We further add an entropy regularization term $-\lambda \cdot \mathbb{E}_{p_\theta(\sigma_t|X_t)}\left[\log p_\theta(\sigma_t|X_t)\right]$ to each of $\log p_\theta(X_{t-1}|X_t)$. Using the same REINFORCE and Monte-Carlo trick, we obtain

$$\nabla_\theta \left(-\lambda \cdot \mathbb{E}_{p_\theta(\sigma_t|X_t)}\Big[\log p_\theta(\sigma_t|X_t)\Big]\right) \approx \sum_{n=1}^{N} -\lambda \log p_\theta\left(\sigma_t^{(n)}|X_t\right) \nabla_\theta \log p_\theta\left(\sigma_t^{(n)}|X_t\right),$$

where $\sigma_t^{(1)}, \ldots, \sigma_t^{(N)} \sim p_\theta(\sigma_t|X_t)$. Therefore, we have

$$\begin{aligned}
&\nabla_\theta \left(\log p_\theta(X_{t-1}|X_t) - \lambda \cdot \mathbb{E}_{p_\theta(\sigma_t|X_t)}\Big[\log p_\theta(\sigma_t|X_t)\Big]\right) \\
&\approx \sum_{n=1}^{N} \left(\underbrace{\frac{p\left(X_{t-1}|X_t, \sigma_t^{(n)}\right)}{\sum_{m=1}^{N} p\left(X_{t-1}|X_t, \sigma_t^{(m)}\right)} - \lambda \log p_\theta\left(\sigma_t^{(n)}|X_t\right)}_{\texttt{weight}} \right) \cdot \nabla_\theta \log p_\theta\left(\sigma_t^{(n)}|X_t\right), \qquad (18)
\end{aligned}$$

where $\sigma_t^{(1)}, \ldots, \sigma_t^{(N)} \sim p_\theta(\sigma_t|X_t)$. We then substitute in

$$p\left(X_{t-1}|X_t, \sigma_t^{(n)}\right) = \mathcal{N}\left(X_{t-1}|Q_{\sigma_t^{(n)}} X_t, I\right)$$

for all $n \in [N]$. Finally, we also subtract the exponential moving average `weight` as a control variate for variance reduction, where the exponential moving average is given by $\texttt{ema} \leftarrow \texttt{ema\_rate} \cdot \texttt{ema} + (1 - \texttt{ema\_rate}) \cdot \texttt{weight}$ for each gradient descent step.

E.4 Training Details and Architecture Hyperparameters

Hardware. The Jigsaw Puzzle and Sort 4-Digit MNIST Numbers experiments are trained and evaluated on the NVIDIA A40 GPU. The TSP experiments are trained and evaluated on the NVIDIA A40 and A100 GPU.

Jigsaw Puzzle. For the Jigsaw Puzzle experiments, we use the AdamW optimizer [26] with weight decay 1e-2, $\varepsilon =$ 1e-9, and $\beta = (0.9, 0.98)$. We use the Noam learning rate scheduler given in [44] with 51,600 warmup steps for Noisy MNIST and 46,000 steps for CIFAR-10. We train for 120 epochs with a batch size of 64. When computing the loss (12), we use Monte-Carlo estimation for the expectation and sample 3 trajectories. For REINFORCE, we sampled 10 times for the Monte-Carlo estimation in (18), and we used an entropy regularization rate $\lambda = 0.05$ and an `ema_rate` of 0.995. The neural network architecture and related hyperparameters are given in Table 7. The denoising schedules, with riffle shuffles as the forward process and GPL as the reverse process, are give in Table 8. For beam search, we use a beam size of 200 when decoding from GPL, and we use a beam size of 20 when decoding along the diffusion denoising schedule.

Layer	Details
Convolution	Output channels 32, kernel size 3, padding 1, stride 1
Batch Normalization	–
ReLU	–
Max-pooling	Pooling 2
Fully-connected	Output dimension $(\texttt{dim\_after\_conv} + 128)/2$
ReLU	–
Fully-connected	Output dimension 128
Transformer encoder	7 layers, 8 heads, model dimension (d_{model}) 128, feed-forward dimension 512, dropout 0.1

Table 7: Jigsaw puzzle neural network architecture and hyperparameters.

Number of patches per side	Denoising schedule
2×2	$[0, 2, 7]$
3×3	$[0, 3, 5, 9]$
4×4	$[0, 4, 6, 10]$
5×5	$[0, 5, 7, 11]$
6×6	$[0, 6, 8, 12]$

Table 8: Denoising schedules for the Jigsaw Puzzle task, where we use riffle shuffle in the forward process and GPL in the revserse process.

Sort 4-Digit MNIST Numbers. For the task of sorting 4-digit MNIST numbers, we use the exact training and beam search setup as the Jigsaw Puzzle, except that we do not need to use REINFORCE. The neural network architecture is given in Table 9. The denoising schedules, with riffle shuffles as the forward process and GPL as the reverse process, are give in Table 10.

Layer	Details
Convolution	Output channels 32, kernel size 5, padding 2, stride 1
Batch Normalization	–
ReLU	–
Max-pooling	Pooling 2
Convolution	Output channels 64, kernel size 5, padding 2, stride 1
Batch Normalization	–
ReLU	–
Max-pooling	Pooling 2
Fully-connected	Output dimension $(\texttt{dim\_after\_conv} + 128)/2$
ReLU	–
Fully-connected	Output dimension 128
Transformer encoder	7 layers, 8 heads, model dimension (d_{model}) 128, feed-forward dimension 512, dropout 0.1

Table 9: Sort 4-digit MNIST numbers neural network architecture and hyperparameters.

Sequence Length n	Denoising schedule
3	$[0, 2, 7]$
5	$[0, 2, 8]$
7	$[0, 3, 8]$
9	$[0, 3, 5, 9]$
15	$[0, 4, 7, 10]$
32	$[0, 5, 7, 12]$
52	$[0, 5, 6, 7, 10, 13]$
100	$[0, 8, 10, 15]$

Table 10: Denoising schedules for the Sort 4-Digit MNIST Numbers task, where we use riffle shuffle in the forward process and GPL in the revserse process.

TSP. For solving the TSP, we perform supervised learning to train our SymmetricDiffusers to solve the TSP. Let σ^* be an optimal permutation, and let X_0 be the list of nodes ordered by σ^*. We note that any cyclic shift of X_0 is also optimal. Thus, for simplicity and without loss of generality, we always assume $\sigma^*(1) = 1$. In the forward process of SymmetricDiffusers, we only shuffle the second to the n^{th} node (or component). In the reverse process, we mask certain parameters of the reverse distribution so that we will always sample a permutation with $\sigma_t(1) = 1$.

The architecture details are slightly different for TSP, since we need to input both node and edge features into our network. Denote by X_t the ordered list of nodes at time t. We obtain $Y_t \in \mathbb{R}^{n \times d_{\text{model}}}$ as in Eq. (15), where $\texttt{encoder}_\theta$ is now a sinusoidal embedding of the 2D coordinates. Let $D_t \in \mathbb{R}^{n \times n}$ be the matrix representing the pairwise distances of points in X_t, respecting the order in X_t. Let $E_t \in \mathbb{R}^{\binom{n}{2}}$ be the flattened vector of the upper triangular part of D_t. We also apply sinusoidal embedding to E_t and add $\texttt{time\_embd}(t)$ to it. We call the result $F_t \in \mathbb{R}^{\binom{n}{2} \times d_{\text{model}}}$.

Now, instead of applying the usual transformer encoder with self-attentions, we alternate between cross-attentions and self-attentions. For cross-attention layers, we use the node representations from the previous layer as the query, and we always use $K = V = F_t$. We also apply an attention mask to the cross-attention, so that each node will only attend to edges that it is incident with. For self-attention layers, we always use the node representations from the previous layer as input. We always use an even number of layers, with the first layer being a cross-attention layer, and the last layer being a self-attention layer structured to produce the required parameters for the reverse distribution as illustrated in Appendix B. For hyperparameters, we use 16 alternating layers, 8 attention heads, $d_{\text{model}} = 256$, feed-forward hidden dimension 1024, and dropout rate 0.1.

For training details on the TSP-20 task, we use the AdamW optimizer [26] with weight decay 1e-4, $\varepsilon =$ 1e-8, and $\beta = (0.9, 0.999)$. We use the cosine annealing learning rate scheduler starting from 2e-4 and ending at 0. We train for 50 epochs with a batch size of 512. When computing the loss (12), we use Monte-Carlo estimation for the expectation and sample 1 trajectory. We use a denoising schedule of $[0, 4, 5, 7]$, with riffle shuffles as the forward process and GPL as the reverse process. Finally, we use beam search for decoding, and we use a beam size of 256 both when decoding from GPL and decoding along the denoising schedule.

E.5 Baselines Implementation Details

Gumbel-Sinkhorn Network. We have re-implemented the Gumbel-Sinkhorn Network [29] for application on jigsaw puzzles, following the implementations provided in the official repository[2]. To ensure a fair comparison, we conducted a thorough grid search of the model's hyper-parameters. The parameters included in our search space are as follows,

Hyperparameter	**Values**
Learning Rate (lr)	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
Batch Size	$\{50\}$
Hidden Channels	$\{64, 128\}$
Kernel Size	$\{3, 5\}$
τ	$\{0.2, 0.5, 1, 2, 5\}$
Number of Sinkhorn Iterations (n_sink_iter)	$\{20\}$
Number of Samples	$\{10\}$

Table 11: Hyperparameter Search Space for the Gumbel-Sinkhorn Network

Diffsort & Error-free Diffsort We have implemented two differentiable sorting networks from the official repository[3] specific to error-free diffsort. For sorting 4-digit MNIST images, error-free diffsort employs TransformerL as its backbone, with detailed hyperparameters listed in Table 12. Conversely, Diffsort uses a CNN as its backbone, with a learning rate set to $10^{-3.5}$; the relevant hyperparameters are outlined in Table 13.

For jigsaw puzzle tasks, error-free diffsort continues to utilize a transformer, whereas Diffsort employs a CNN. For other configurations, we align the settings with those of tasks having similar sequence lengths in the 4-digit MNIST sorting task. For instance, for 3×3 puzzles, we apply the same configuration as used for sorting tasks with a sequence length of 9.

TSP. For the baselines for TSP, we first have 4 traditional operations research solvers. Gurobi [14] and Concorde [1] are known as exact solvers, while LKH-3 [15] is a strong heuristic and 2-Opt [25]

[2] `https://github.com/google/gumbel_sinkhorn`

[3] `https://github.com/jungtaekkim/error-free-differentiable-swap-functions`

Sequence Length	Steepness	Sorting Network	Loss Weight	Learning Rate
3	10	odd even	1.00	10^{-4}
5	26	odd even	1.00	10^{-4}
7	31	odd even	1.00	10^{-4}
9	34	odd even	1.00	10^{-4}
15	25	odd even	0.10	10^{-4}
32	124	odd even	0.10	10^{-4}
52	130	bitonic	0.10	$10^{-3.5}$
100	140	bitonic	0.10	$10^{-3.5}$

Table 12: Hyperparameters for Error-Free Diffsort on Sorting 4-Digit MNIST Numbers

Sequence Length	Steepness	Sorting Network
3	6	odd even
5	20	odd even
7	29	odd even
9	32	odd even
15	25	odd even
32	25	bitonic
52	25	bitonic
100	25	bitonic

Table 13: Hyperparameters for Diffsort on Sorting 4-Digit MNIST Numbers

is a weak heuristic. For LKH-3, we used 500 trials, and for 2-Opt, we used 5 random initial guesses with seed 42.

For the GCN model[18], we utilized the official repository[4] and adhered closely to its default configuration for the TSP-20 dataset. For DIFUSCO[43], we sourced it from its official repository[5] and followed the recommended configuration of TSP-50 dataset, with a minor adjustment in the batch size. We increased the batch size to 512 to accelerate the training process. For fair comparison, we also remove the post-processing heuristics in both models during the evaluation.

F Limitations

Despite the success of this method on various tasks, the model presented in this paper still requires a time-space complexity of $\mathcal{O}(n^2)$ due to its reliance on the parametric representation of GPL and the backbone of transformer attention layers. This complexity poses a significant challenge in scaling up to applications involving larger symmetric groups or Lie groups.

[4] `https://github.com/chaitjo/graph-convnet-tsp`

[5] `https://github.com/Edward-Sun/DIFUSCO`

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and Section 1 accurately summarize the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of the work in Appendix F.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide complete proof for Proposition 1 and Proposition 2 in Appendix D.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In Appendix E, we fully disclose all information to reproduce our experimental results, including dataset preparation, training details, and choices of hyperparameters as well as baselines' implementation details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We've included codes to reproduce the main results in the supplemental material. We also attach a detailed README file that provides sufficient instructions.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: All experimental settings and details are specified in Appendix E.4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All reported experimental results are averaged over at least three runs with different random seeds.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide information on the computation resources used for our experiments in Appendix E.4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We preserve anonymity with the NeurIPS Codes of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited the original paper of our reference code and datasets.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.