1-Department of Computer Science, Purdue University, West Lafayette, IN, USA

2-Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA

Discrete Tree Flows via Tree-Structured Permutations

Mai Elkady[1]*, Jim Lim[2]* & David I.Inouye[2]

PURDUE UNIVERSITY

Setting

Motivation
Discrete data is abundant in many forms and domains, like DNA sequences, medical records, molecular structure and text. Thus, analyzing discrete data by modeling and inferring its distribution is crucial in a multitude of applications.

Background: Discrete Flows
While continuous flow models are based on the continuous change of variables, i.e., $P_x(x) = Q_z(f(x)) \det \left| \frac{df(x)}{dx} \right|$, where f is an invertible model, *discrete flows* are based on the discrete change of variables, i.e., $P_x(x) = Q_z(\sigma(x))$, where σ is a **permutation** (i.e., an invertible discrete transform) and the Jacobian term is 1 because permutations cannot change volume.

Gap
Prior discrete flow works focus on **discrete approximations** via projecting to continuous space or using a straight-through gradient estimator. However, prior models still suffer from the following limitations:

- Gradients of discrete functions are **undefined or zero** therefore conventional optimization cannot be applied.
- Discrete functions can only be approximated with pseudo gradients and backpropagation is **burdensome** compared to alternative discrete algorithms.

Approach
Our proposed approach is based on decision trees. Our Tree-Structured Permutations (TSP) encode permutations at each node, and are inevitable if a simple constraint is satisfied.

Tree Structured Permutations (TSPs)

A *Tree-Structured Permutation* (TSP) is a binary decision tree where each node contains both a permutation $\pi_{\mathcal{N}}$ and split information.

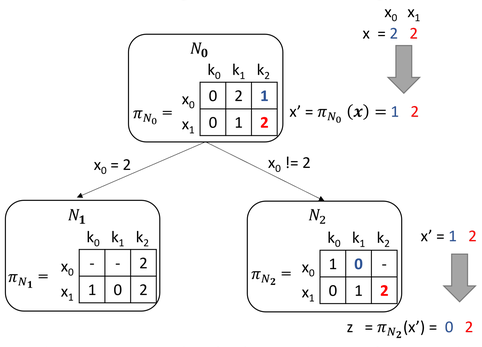

Figure 2: This is an example of a forward pass of a TSP where π_N's represent independent $d \times k$ matrix permutation. On node N_0 the split is on the 0^{th} feature x_0 and on the category value of 2. π_{N_0} permutes the input [2, 2] to [1, 2] and since this input's $x_0! = 2$ after the permutation it goes to the right child where it undergoes another permutation.

Learning Objective: We seek the TSP $\sigma_{\mathcal{T}}$ that minimizes the negative log-likelihood assuming the prior distribution Q_z is *independent*:

$$\arg\min_{\sigma_{\mathcal{T}}} \min_{Q_z \in \mathbb{Q}_{\text{Ind}}} -\frac{1}{n} \sum_{i=1}^{n} \log Q_z(\sigma_{\mathcal{T}}(x_i)) \,.$$

Algorithm Illustration

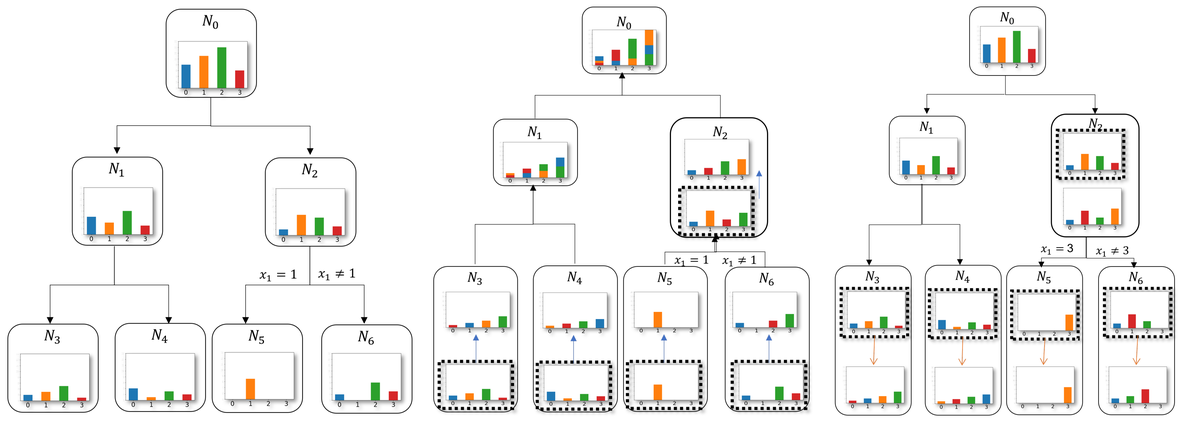

Figure 1: Illustration of our two pass algorithm used to learn the optimal node permutations. The leftmost diagram demonstrates a constructed TSP where the bar plots represents counts of the categories (0, 1, 2, 3). We show just x_1's counts for simplicity, but the same idea applies to other features. In the middle, we demonstrate the operation of pass 1. The dashed lines around borders of the bar plots indicate that these are intermediate steps. The blue arrows indicate that we are learning and applying the local permutations for x_1 at the corresponding node. On the right, we demonstrate the resulting tree after both passes. The orange arrows indicate the new node permutations.

Stage 1: Learn Tree Structure

We use a standard greedy decision tree algorithm for learning the tree structure and test two different splitting criteria:

- **DTF$_{RND}$ (Baseline):** based on randomly choosing a split feature and a split value for each node.
- **DTF$_{GLP}$ (Novel):** based on the *greedy local permutation* (GLP) heuristic that chooses the best split feature and split value for decreasing the NLL when applying a hypothesized optimal permutation.

Stage 2: Learn Node Permutations

We propose a two pass algorithm for learning node permutations:

- **Pass 1:** The algorithm starts at the leaf nodes and traverses the tree in a **bottom-up fashion**. At each leaf node, the algorithm sorts the local node category counts of each feature in ascending order. At internal nodes, the local counts from children are added and then sorted again if needed.
- **Pass 2:** Using the permutations associated with the sorting in the first pass, we traverse back down the tree in a **top-down fashion** to construct a new equivalent TSP that is guaranteed to be optimal (i.e., *rank consistent*).

Therefore, the two pass algorithm will be create a **rank consistent** tree equivalent TSP which is **optimal** in terms of NLL.

Experimental Results

	AF	BF	DDF	DTF$_{GLP}$	DTF$_{RND}$
			Mushroom		
NLL	24.87 (± 2.28)	23.02 (± 2.3)	19.18 (± 3.48)	**14.15 (± 2.44)**	16.66 (± 2.98)
TTC	29.3 (± 2.0)	20.9 (± 2.7)	175.8 (± 1.9)	9.9 (± 0.2)	**0.5 (± 0.0)**
TTG	7.7 (± 1.0)	**6.0 (± 1.3)**	75.2 (± 1.0)	NA	NA
			MNIST		
NLL	206.014 (± 0.32)	205.94 (± 0.26)	**144.78 (± 10.52)**	177.75 (± 0.56)	187.44 (± 1.17)
TTC	12104.6 (± 359.2)	3290.5 (± 13.3)	2909.3 (± 45.4)	5213.7 (± 204.9)	**105.6 (± 0.1)**
TTG	**305.6 (± 31.4)**	308.7 (± 4.1)	334.3 (± 8.7)	NA	NA
			Genetic		
NLL	490.55 (± 0.69)	471.54 (± 1.87)	446.86 (± 8.64)	**437.19 (± 1.02)**	470.9 (± 6.1)
TTC	834.0(± 2.1)	251.6 (± 0.5)	209.4 (± 0.6)	411.5 (± 2.3)	**5.9 (± 0.0)**
TTG	**23.8 (± 0.9)**	38.0 (± 5.4)	29.5 (±1.1)	NA	NA

(a) Real-world dataset results.

	AF	BF	DDF	DTF$_{GLP}$
			8Gaussian	
NLL	6.92 (± 0.06)	7.21 (± 0.09)	**6.42 (± 0.03)**	6.5 (± 0.03)
TTC	155.9 (± 2.2)	231.6 (± 5.2)	119.8 (± 0.8)	**7.3 (± 0.1)**
TTG	42.2 (± 3.5)	135.8 (± 0.2)	79.7 (± 0.8)	NA
			COP-H	
NLL	1.53 (± 0.02)	1.47 (± 0.06)	1.46 (± 0.1)	**1.33 (± 0.02)**
TTC	10.7 (± 0.2)	13.2 (± 0.2)	58.1 (± 1.0)	**≤0.1 (± 0.0)**
TTG	14.6 (± 0.4)	20.9 (±0.3)	48.8 (±0.9)	NA
			COP-M	
NLL	1.76 (± 0.1)	1.62 (± 0.05)	1.51 (± 0.16)	**1.4 (± 0.02)**
TTC	10.6 (± 0.02)	13.3 (± 0.06)	77.9 (± 1.8)	**≤0.1(± 0.0)**
TTG	14.6 (± 0.43)	20.8 (± 0.8)	66.9 (±0.5)	NA
			COP-W	
NLL	2.42 (± 0.02)	2.35 (± 0.03)	2.29 (± 0.07)	**2.22 (± 0.02)**
TTC	10.5 (± 0.01)	13.2 (± 0.1)	77.3 (± 1.7)	**≤0.1 (± 0.0)**
TTG	13.9(± 0.2)	19.2 (± 0.2)	67.5(± 0.1)	NA

(b) Synethetic dataset results.

Figure 3: TTC: Training time in secs on CPU, TTG: Training time in secs on GPU, NLL: Negative Loglikelihood, AF: Autoregressive Flows, BF: Bipartite Flows, DDF: Discrete Denoising Flows. Yellow highlights: best performance, blue highlights: second best.

On synthetic datasets, our model always have a **better training time** on CPU with **comparable or better negative loglikelihoods**. On high dimensional real-world datasets, our model with random splitting is **fastest** on CPU, while the GLP gives **comparable** NLLs

Properties of TSPs

Key Definitions

- *Tree Equivalence:* Two TSP tree structures are considered tree equivalent if and only if they have the same graph structure (i.e. same nodes and edges) and the same data path configuration (i.e. nodal path of each individual data input).
- *Rank Consistency* A TSP tree is rank consistent if and only if there exists an independent permutation π such that the data categorical counts of any node (after applying all ancestral nodal permutations) is sorted in ascending order respective to its categorical values.

Invertibility
To ensure our TSPs are invertible (and thus applicable to discrete flows), we prove that invertiblity is guaranteed if all node permutations $\pi_{\mathcal{N}}$ do not permute configurations that are outside of the node's domain, i.e., $\pi_{\mathcal{N}}(x) = x, \forall x \notin \mathcal{D}(\mathcal{N})$.

Optimality of Rank Consistent TSPs
We prove that a rank consistent TSP produces the optimal NLL among TSPs that are tree equivalent and, ultimately, the optimal solution to the objective function.

Universal Model
We demonstrate and prove that a sequential composition of TSPs can produce a universal permutation.

Tractability
We restrict to independent feature-wise permutation which allows each feature to be permuted independently of other features. Thus, we define a computational tractable and generalizable model.

Equivalence Relation of Tree Equivalence
We prove that all tree equivalent TSPs also hold equivalence relation (i.e. holds reflexive, symmetric, and transitive properties).

Conclusion and Discussion

Conclusions

- Our results demonstrate that DTF outperforms prior approaches in terms of NLL while being substantially faster for most experiments
- We lay the groundwork for developing practical and effective discrete flows using decision tree algorithms sidestepping problems with back-propagation

Limitations

- While the permutations are guaranteed to be optimal, the tree structure is not since the tree is grown greedily
- Our splitting is axis aligned, and not complex
- As with previous models, large values of categories are challenging to handle
- Our approach works best for tabular data and extending to image or text is non-trivial

Future directions

- Upgrade model to be capable of more complex splits, potentially relying on Neural Networks to power the splitting function, this would in turn allow us to extend the model beyond tabular data