A Appendix

A.1 Proof of Proposition 1

Proof. Given a dataset X, pick an element $x \in X$. We consider all possible Ω clusters X_L^ω in $\mathbb{C}(X)_x$. Given X_L^ω, then X_R^ω is fixed so as to satisfy $X_L^\omega \bigcup X_R^\omega = X$ and $X_L^\omega \bigcap X_R^\omega = \emptyset$. We want to show that the partition function $Z(X)$ can be written recursively in terms of $Z(X_L^\omega)$ and $Z(X_R^\omega)$.

The partition function is defined as the sum of the energies of all possible hierarchical clusterings $\mathcal{H}_X = \{\mathtt{H}^m\}_{m=1}^M$,

$$Z(X) = \sum_{m=1}^{M} \phi(\mathtt{H}^m(X)) = \sum_{m=1}^{M} \psi(X_L^m, X_R^m)\ \phi(\mathtt{H}^m(X_L^m))\ \phi(\mathtt{H}^m(X_R^m)) \tag{8}$$

where $X_L^m \bigcup X_R^m = X$, $X_L^m \bigcap X_R^m = \emptyset$. Also, $\mathtt{H}^m(X_L^m)$ and $\mathtt{H}^m(X_R^m)$ are the sub-hierarchies in $\mathtt{H}^m$ that are rooted at X_L^m and X_R^m, respectively. Next, we rewrite Eq. 8 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$ [4],

$$Z(X) = \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega) \sum_{j=1}^{J} \phi(\mathtt{H}^j(X_L^\omega)) \sum_{k=1}^{K} \phi(\mathtt{H}^k(X_R^\omega)) = \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z(X_L^\omega)\ Z(X_R^\omega) \tag{9}$$

with $M = \Omega \cdot J \cdot K$, $J = (2|X_L^\omega| - 3)!!$, and $K = (2|X_R^\omega| - 3)!!$ for a full trellis. Thus, $Z(X)$ of a cluster X can be written recursively in terms of the partition function of the sub-clusters of X [5].

□

A.2 Proof of Theorem 3

Proof. We want to show that drawing samples of trees using Algorithm 3 gives samples from $P(\mathtt{H}|X)$. To do this, we show that the probability of a tree can be re-written as the product of probabilities of sampling each split in the structure. This then directly corresponds to the top-down sampling procedure in Algorithm 3.

Recall from Definition 2 we have:

$$P(\mathtt{H}|X) = \frac{1}{Z(X)} \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} \psi(X_L, X_R) \tag{10}$$

We can equivalently write this as:

$$P(\mathtt{H}|X) = \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} \frac{1}{Z(X_L \cup X_R)} \cdot \psi(X_L, X_R) \cdot Z(X_L) \cdot Z(X_R) \tag{11}$$

To understand why this can be written this way, observe that for internal nodes the $Z(X_L)$ and $Z(X_R)$ terms will be cancelled out by corresponding terms in the product for the children of X_L or X_R. To see this we can write out the product for three pairs of nodes X_L, X_R and their children X_{LL}, X_{LR} and X_{RL} and X_{RR} respectively:

$$\frac{1}{Z(X_p)}\psi(X_L, X_R)\ Z(X_L)\ Z(X_R) \cdot \frac{1}{Z(X_L)}\psi(X_{LL}, X_{LR})\ Z(X_{LL})\ Z(X_{LR}) \cdot \frac{1}{Z(X_R)}\psi(X_{RL}, X_{RR})\ Z(X_{RL})\ Z(X_{RR}) \tag{12}$$

Recall that for the pair of siblings that are the children of the root, the $\frac{1}{Z(X_L \cup X_R)}$ term will not be cancelled out and corresponds exactly to $\frac{1}{Z(X)}$.

Next, we observe that Eq. 11 can be re-written in terms of Equation 3 which defines $p(X_L|X_L \cup X_R)$:

$$P(\mathtt{H}|X) = \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} p(X_L|X_L \cup X_R) \tag{13}$$

Algorithm 3 applies Eq. 3 recursively in a top-down manner using a series of splits which have a probability that directly corresponds to the product of terms in Eq. 13.

□

[4] The cluster trellis provides an exact solution conditioned on the fact that the domain of the linkage function is the set of pairs of clusters, and not pairs of trees.

[5] Note that for each singleton x_i, we have $Z(x_i) = 1$.

A.3 Proof of Lower Bound on Number of Trees

The number of trees on N leaves is given exactly by $\frac{\prod_{i=1}^{N-1} m_i}{(N-1)!} \prod_{i=2}^{N} \binom{i}{2}$, where m_i is the number of internal nodes in the subtree rooted at node i (Boyles and Welling, 2012). Since $\prod_{i=2}^{N} \binom{i}{2} = \frac{N!^2}{N*2^{N-1}}$, this makes the number of trees on N leaves $\frac{\prod_{i=1}^{N-1} m_i}{(N-1)!} \frac{N!^2}{N*2^{N-1}} = \frac{\prod_{i=1}^{N-1} m_i * N*N!}{N*2^{N-1}} = \frac{\prod_{i=1}^{N-1} m_i * N!}{2^{N-1}}$. The smallest conceivable value for $\prod_{i=1}^{N-1} m_i = \omega(N)$, which gives us the bound on the number of trees to be $\omega(NN!/2^{N-1})$, as desired.

Note that this is a loose lower-bound, and that it could be improved upon as follows: say a hierarchical clustering is a *caterpillar clustering* is every internal node in the underlying tree has two children and the set associated with one of those children as size one. There are $n!/2$ caterpillar clustering. To see this, note that the ith level (where the root is level 1) of a caterpillar clustering has exactly one leaf for $i = 2, \ldots, n-1$. There are $n(n-1)\ldots 3 = n!/2$ choices for the corresponding singleton sets.

Note, however, that there is a closed form expression for the exact number of unordered hierarchies given by $a(N) = (2N-3)!!$, with n the number of singletons (see (Callan, 2009; Dale and Moon, 1993) for more details and proof).

A.4 Correctness Proof of Marginal Algorithms

A.4.1 Sub-Hierarchy Marginal

For a given sub-hierarchy rooted at X_i, i.e., $\mathtt{H}_i \in \mathcal{H}(X_i)$, the marginal probability is defined as $P(\mathtt{H}_i|X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} P(\mathtt{H}|X)$, where $A(\mathtt{H}_i) = \{\mathtt{H} : \mathtt{H} \in \mathcal{H}(X) \wedge \mathtt{H}_i \subset \mathtt{H}\}$, and $\mathtt{H}_i \subset \mathtt{H}$ indicates that $\mathtt{H}_i$ is a subtree of $\mathtt{H}$. We can rewrite $\sum_{\mathtt{H} \in A(\mathtt{H}_i)} P(\mathtt{H}|X)$ as $\sum_{\mathtt{H} \in A(\mathtt{H}_i)} \phi(\mathtt{H}(X))/Z$, which gives us:

$$P(\mathtt{H}_i|X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} P(\mathtt{H}|X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} \frac{\phi(\mathtt{H}(X))}{Z} = \frac{Z_{\mathtt{H}_i}(X)}{Z} \tag{14}$$

where $Z_{\mathtt{H}_i}(X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} \phi(\mathtt{H}(X))$, the sum of potential values for all the hierarchies containing the sub-hierarchy $\mathtt{H}_i$. This gives us

$$\begin{aligned} Z_{\mathtt{H}_i}(X) &= \sum_{m=1}^{|A(\mathtt{H}_i)|} \phi(\mathtt{H}^m(X)) \\ &= \sum_{m=1}^{|A(\mathtt{H}_i)|} \psi(X_L^m, X_R^m)\ \phi(\mathtt{H}^m(X_L^m))\ \phi(\mathtt{H}^m(X_R^m)) \end{aligned} \tag{15}$$

where $X_L^m \bigcup X_R^m = X$, $X_L^m \bigcap X_R^m = \emptyset$. Also, $\mathtt{H}^m(X_L^m)$ and $\mathtt{H}^m(X_R^m)$ are the sub-hierarchies in $\mathtt{H}^m$ that are rooted at X_L^m and X_R^m, respectively. Next, we rewrite Eq. 15 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$. Note that $\mathtt{H}_i \subset \mathtt{H}$, implies $X_i \subseteq X_L$ or $X_i \subseteq X_R$. Assume W.L.O.G. that $X_i \subseteq X_L$.

$$\begin{aligned} Z_{\mathtt{H}_i}(X) &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ \sum_{j=1}^{J} \phi(\mathtt{H}^j(X_L^\omega))\ \sum_{k=1}^{K} \phi(\mathtt{H}^k(X_R^\omega)) \\ &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z_{\mathtt{H}_i}(X_L^\omega)\ Z(X_R^\omega) \end{aligned} \tag{16}$$

with $|A(\mathtt{H}_i)| = \Omega \cdot J \cdot K$, $J = |\{\mathcal{H}(X_L) : X_i \subseteq X_L\}|$, $K = |\{\mathcal{H}(X_R)\}|$ and setting $Z_{\mathtt{H}_i}(X_i) = \phi(\mathtt{H}(X_i))$.

A.4.2 Subset Marginal

For a given cluster X_i, the marginal probability is defined as $P(X_i|X) = \sum_{\mathtt{H} \in A(X_i)} P(\mathtt{H}|X)$, where $A(X_i) = \{\mathtt{H} : \mathtt{H} \in \mathcal{H}(X) \wedge X_i \subset \mathtt{H}\}$, and $X_i \subset \mathtt{H}$ indicates that cluster X_i is contained in $\mathtt{H}$. We can rewrite $\sum_{\mathtt{H} \in A(X_i)} P(\mathtt{H}|X)$ as $\sum_{\mathtt{H} \in A(X_i)} \phi(\mathtt{H}(X))/Z$, which gives us:

$$P(X_i|X) = \sum_{\mathtt{H} \in A(X_i)} P(\mathtt{H}|X) = \sum_{\mathtt{H} \in A(X_i)} \frac{\phi(\mathtt{H}(X))}{Z} = \frac{Z_{X_i}(X)}{Z} \tag{17}$$

where $Z_{X_i}(X) = \sum_{\mathtt{H}\in A(X_i)}\phi(\mathtt{H}(X))$, the sum of potential values for all the hierarchies containing the cluster X_i. This gives us

$$Z_{X_i}(X) = \sum_{m=1}^{|A(X_i)|} \phi(\mathtt{H}^m(X)) = \sum_{m=1}^{|A(X_i)|} \psi(X_L^m, X_R^m)\ \phi(\mathtt{H}^m(X_L^m))\ \phi(\mathtt{H}^m(X_R^m)) \tag{18}$$

where $X_L^m \bigcup X_R^m = X$, $X_L^m \bigcap X_R^m = \emptyset$. Also, $\mathtt{H}^m(X_L^m)$ and $\mathtt{H}^m(X_R^m)$ are the sub-hierarchies in $\mathtt{H}^m$ that are rooted at X_L^m and X_R^m, respectively. Next, we rewrite Eq. 18 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$. Note that $X_i \subset \mathtt{H}$, implies $X_i \subseteq X_L$ or $X_i \subseteq X_R$. Assume W.L.O.G. that $X_i \subseteq X_L$.

$$\begin{aligned} Z_{X_i}(X) &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega) \sum_{j=1}^{J} \phi(\mathtt{H}^j(X_L^\omega)) \sum_{k=1}^{K} \phi(\mathtt{H}^k(X_R^\omega)) \\ &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z_{X_i}(X_L^\omega)\ Z(X_R^\omega) \end{aligned} \tag{19}$$

with $|A(\mathtt{H}_i)| = \Omega \cdot J \cdot K$, $J = |\{\mathcal{H}(X_L) : X_i \subseteq X_L\}|$, $K = |\{\mathcal{H}(X_R)\}|$, and setting $Z_{X_i}(X_i) = Z(X_i)$.

A.5 Proof of MAP Time Complexity

The MAP tree is computed for each node in the trellis, and due to the order of computation, at the time of computation for node i, the MAP trees for all nodes in the subtrellis rooted at node i have already been computed. Therefore, the MAP tree for a node with i elements can be computed in 2^i steps (given the pre-computed partition functions for each of the node's descendants), since the number of nodes for the trellis rooted at node i (with i elements) corresponds to the powerset of i. There are $\binom{n}{i}$ nodes of size i, making the total computation $\sum_{i=1}^{N} 2^i \binom{N}{i} = 3^N - 1$.

A.6 Proof of Proposition 2

Proof. We proceed in a similar way as detailed in Appendix § A.1 , as follows. Given a dataset X, pick an element $x \in X$. We consider all possible Ω clusters X_L^ω in $\mathbb{C}(X)_x$. Given X_L^ω, then X_R^ω is fixed so as to satisfy $X_L^\omega \bigcup X_R^\omega = X$ and $X_L^\omega \bigcap X_R^\omega = \emptyset$. We want to show that the MAP clustering $\phi(\mathtt{H}^*(X))$ can be computed recursively in terms of $\phi(\mathtt{H}^*(X_L^\omega))$ and $\phi(\mathtt{H}^*(X_R^\omega))$.

The MAP value is defined as the energy of the clustering with maximal energy ϕ among all possible hierarchical clusterings $\mathcal{H}_X = \{\mathtt{H}^m\}_{m=1}^M$,

$$\begin{aligned} \phi(\mathtt{H}^*(X)) &= \max_{m\in M} \phi(\mathtt{H}^m(X)) \\ &= \max_{m\in M} \psi(X_L^m, X_R^m)\ \phi(\mathtt{H}^m(X_L^m))\ \phi(\mathtt{H}^m(X_R^m)) \end{aligned} \tag{20}$$

where $X_L^m \bigcup X_R^m = X$, $X_L^m \bigcap X_R^m = \emptyset$. Also, $\mathtt{H}^m(X_L^m)$ and $\mathtt{H}^m(X_R^m)$ are the sub-hierarchies in $\mathtt{H}^m$ that are rooted at X_L^m and X_R^m, respectively. As mentioned earlier, the cluster trellis provides an exact MAP solution conditioned on the fact that the domain of the linkage function is the set of pairs of clusters, and not pairs of trees. Thus, we can rewrite Eq. 20 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$, as follows

$$\begin{aligned} \phi(\mathtt{H}^*(X)) &= \max_{\omega\in\Omega} \left(\psi(X_L^\omega, X_R^\omega) \max_{j\in J} \phi(\mathtt{H}^j(X_L^\omega)) \max_{k\in K} \phi(\mathtt{H}^k(X_R^\omega)) \right) \\ &= \max_{\omega\in\Omega} \psi(X_L^\omega, X_R^\omega)\ \phi(\mathtt{H}^*(X_L^\omega))\ \phi(\mathtt{H}^*(X_R^\omega)) \end{aligned} \tag{21}$$

with $M = \Omega \cdot J \cdot K$. Thus, $\phi(\mathtt{H}^*(X))$ of a cluster X can be written recursively in terms of the MAP values of the sub-clusters of X [6].

□

[6] Note that for each singleton x_i, we have $\phi(\mathtt{H}^*(x_i)) = 1$.

A.7 Proofs of Theorem 1 and Corollary 2

The partition function is computed for each node in the trellis, and due to the order of computation, at the time of computation for node i, the partition functions for all nodes in the subtrellis rooted at node i have already been computed. Therefore, the partition function for a node with i elements can be computed in 2^i steps (given the pre-computed partition functions for each of the node's descendants), since the number of nodes for the trellis rooted at node i (with i elements) corresponds to the powerset of i. There are $\binom{N}{i}$ nodes of size i, making the total computation $\sum_{i=1}^{N} 2^i \binom{N}{i} = 3^N - 1$.

In Corollary 2 we state that Algorithm 1 is super-exponentially more efficient than brute force methods that consider every possible hierarchy. Their ratio is

$$r = \frac{(2N-3)!!}{3^N} = \frac{1}{2\sqrt{\pi}} \left(\frac{2}{3}\right)^N \Gamma(N - 1/2) \tag{22}$$

with Γ the gamma function. Thus, r presents a super-exponential growth in terms of N.

A.8 Jet Physics Background

It is natural to represent a jet and the particular clustering history that gave rise to it as a binary tree, where the inner nodes represent each of the unstable particles and the leaves represent the jet constituents. This representation connects jets physics with natural language processing (NLP) and biology, which is exciting and was first suggested in (Louppe et al., 2019).

Jets are among the most common objects produced at the Large Hadron Collider (LHC) at CERN, and a great amount of work has been done to develop techniques for a better treatment and understanding of them, from both an experimental and theoretical point of view. In particular, determining the nature (type) of the initial unstable particle (the root of the binary tree), and its children and grandchildren that gave rise to a specific jet is essential in searches for new physics, as well as precision measurements of our current model of nature, i.e., the Standard Model of particle physics. In this context, it becomes relevant and interesting to study algorithms to cluster the jet constituents (leaves) into a binary tree and metrics to compare them. Being able to improve over the current techniques that attempt to invert the showering process to reconstruct the ground truth-level tree would assist in physics searches at the Large Hadron Collider.

There are software tools called **parton showers**, e.g., PYTHIA, Herwig, Sherpa, that encode a physics model for the simulation of jets that are produced at the LHC. Current algorithms used by the physics community to estimate the clustering history of a jet are domain-specific sequential recombination jet algorithms, called *generalized k_t clustering algorithms* (Cacciari et al., 2008), and they do not use these generative models. These algorithms sequentially cluster the jet constituents by locally choosing the pairing of nodes that minimizes a distance measure. Given a pair of nodes, this measure depends on the angular distance between their momentum vector and the value of this vector in the transverse direction with respect to the collision axis between the incoming beams of protons.

Currently, generative models that implement the parton shower in full physics simulations are implicit models, i.e., they do not admit a tractable density. Extracting additional information that describes the features of the latent process is relevant to study problems where we aim to unify generation and inference, e.g inverting the generative model to estimate the clustering history of a jet. A schematic representation of this approach is shown in Figure 9.

At present, it is very hard to access the joint likelihood in state-of-the-art parton shower generators in full physics simulations. Also, typical implementations of parton showers involve sampling procedures that destroy the analytic control of the joint likelihood. Thus, to aid in machine learning (ML) research for jet physics, a python package for a toy generative model of a parton shower, called Ginkgo, was introduced in (Cranmer et al., 2019b). Ginkgo has a tractable joint likelihood, and is as simple and easy to describe as possible but at the same time captures the essential ingredients of parton shower generators in full physics simulations. Within the analogy between jets and NLP, Ginkgo can be thought of as ground-truth parse trees with a known language model. A python package with a pyro implementation of the model with few software dependencies is publicly available in (Cranmer et al., 2019b).

A.9 Counting Trees

We count the total number of hierarchies[7]. We implement a bottom-up approach and start by assigning a number of trees $N = 1$ to to each cluster of one element. Then, given a parent cluster X_p, we add the contribution N_p^i ($N_p = \sum_i N_p^i$) of each possible pair i of left and right children, $s_{X_p} = \{X_L, X_R\}$, where $X_L \cup X_R = X_p$ and $X_L \cap X_R = \emptyset$. In particular, we obtain

$$N_p^i = N_{X_L}^i \cdot N_{X_R}^i \tag{23}$$

Thus N_p is the number of possible trees of the sub-branch whose root node is X_p. We repeat the process until we reach the cluster of all elements X.

A.10 Runtime Asymptotics Plots

See Figure 10 for a comparison of the number of trees vs the time complexity of the trellis algorithms for finding the partition function, MAP, and marginal values.

A.11 Description of Computer Architecture and Experimental Runtime

When using a MacBook Pro with a 2.5 GHz Intel Core i5 processor with 8GB 1600 MHz DDR3 RAM to compute the MAP for the genetics experiments using the PAM dataset, it takes approximately 15 minutes to complete from start to finish (including data loading and result output). When using this same machine to compute that MAP for Dasgupta cost on the given graph, it takes approximately 4 seconds to complete from start to finish (including data loading and result output).

When using a MacBook Pro with a 2.3 GHz Intel Core i9 processor with 16GB 2400 MHz DDR4 RAM to compute the MAP for the jet physics experiments it takes $5x10^{-2}$, 1.6 and 6.1 seconds to run the trellis on jets with 5, 9 and 10 leaves respectively.

A.12 Sparse Trellis

As mentioned in section 3.1, there are different mappings for the ordering of the leaves of the input trees when building the sparse trellis, and the subset of hierarchies spanned by the trellis depends on this mapping. Specifically, two sub-hierarchies identical under some ordering of the leaves would contribute the same vertices and edges to the trellis. However, this could change by modifying the ordering, e.g. vertex $\{a, b\}$ could turn into vertices $\{a, b\}$ and $\{a, d\}$. Thus, the hierarchies over which the sparse trellis spans depend on the ordering of the leaves of the input trees that we use to build it. We show in Figure 11 the performance of the sparse trellis to calculate the MAP values on a set of 100 Ginkgo jets with 9 leaves. Here we study the sparse trellises for more orderings of the leaves of the input trees than the ones shown in Figure 7.

Next, in Figure 12 we show the number of vertices added to the sparse trellis vs their sparsity (number of trees that they can realize over total possible number of trees). It is interesting to note that the sparsity depends not only on the number of vertices but also on their location in the trellis as well as the edges. Thus, we see that for the same number of vertices, there are different sparsity indices, depending on the building strategy.

[7]This gives a result matching exactly the formula $(2N - 3)!!$

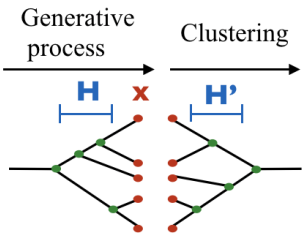

Figure 9: **Schematic representation of the tree structure of a sample jet** generated with Ginkgo and the clustered tree for some clustering algorithm. For a given algorithm, z labels the different variables that determine the latent structure of the tree. The tree leaves x are labeled in red and the inner nodes in green.

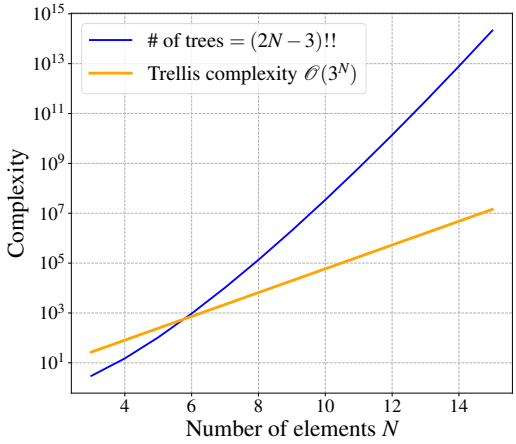

Figure 10: **Comparison of the complexity** of the cluster trellis (orange) and the number of trees (blue) vs the number of elements of a dataset.

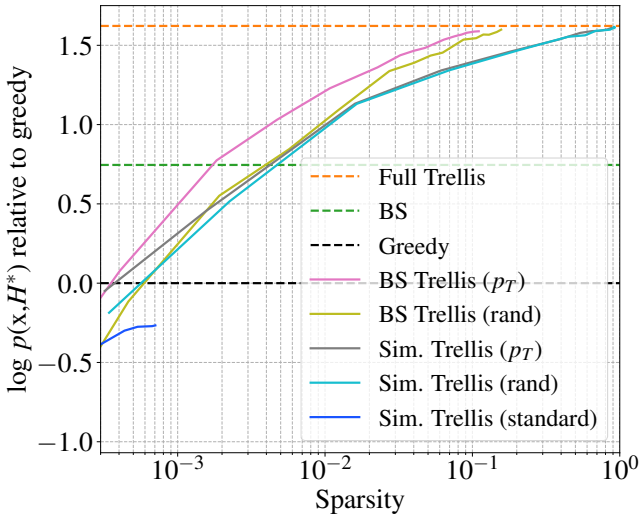

Figure 11: **Trellises MAP hierarchy log likelihood vs their sparsity**. MAP hierarchy log likelihood values are relative to the greedy algorithm. Each value corresponds to the mean over 100 trees of a test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises. We present the trellis obtained by ordering the leaves of the input trees in three different ways. First, in increasing norm of their momentum vector $\vec{p} \in \mathbb{R}^3$ (p_T), see the probabilistic model description of section 4.2 for more details. Second, leaves ordered randomly (rand). Third, leaves ordered by how they are accessed by traversing the trees (standard). Note that in this last case, we only show the Sim. trellis results as the BS trellis spans over sparsity indices values of $\mathcal{O}(10^{-5})$ and has a worse performance. We add the values of the full trellis, beam search and greedy algorithms. The BS trellis approaches the performance of the full one for a smaller sparsity index than the Sim. Trellis.

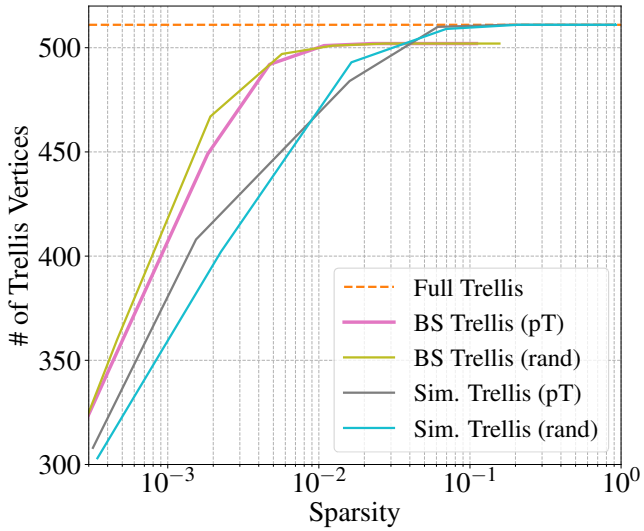

Figure 12: **Trellises number of vertices vs their sparsity**. We show the Simulator (Sim.) and the Beam Search (BS) trellises. The leaves of the input trees are ordered in different ways, as explained in Figure 11. Sim. trellis saturates all the vertices below a sparsity of ~ 0.1 but the performance in Figure 11 keeps increasing. The reason is that we keep adding edges to existing vertices, thus realizing a greater number of trees.

Finally, in Figure 13 we show the MAP hierarchy log likelihood vs algorithms running time on a set of 100 Ginkgo jets with 9 leaves. Also, in both the Simulator (Sim.) and the Beam Search (BS) trellises the nodes have to be initialized, which is done only once for each sparsity index and typically takes between 1 and 10 seconds (depending on the sparsity).

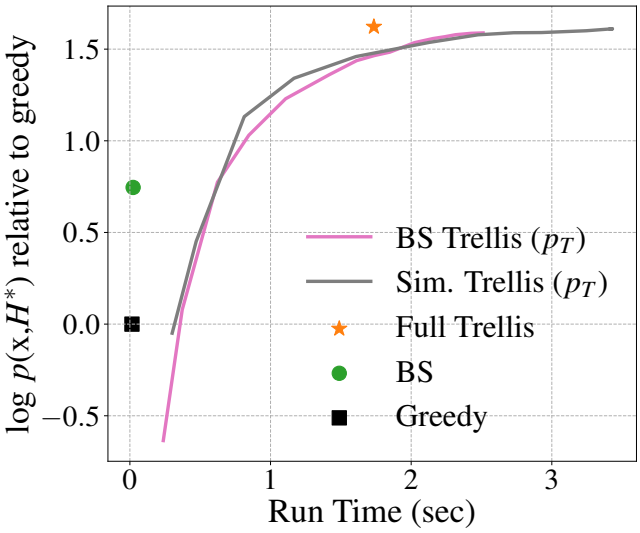

Figure 13: **MAP hierarchy log likelihood vs algorithms running time** on a set of 100 Ginkgo jets with 9 leaves. Each value corresponds to the mean over 100 trees of a test dataset. The difference between the sparse and exact trellises running times is because the exact one is iterative over the nodes and the sparse one is recursive (this was done to optimize memory requirements). We can see that the sparse trellis is faster for low sparsity while the running times are of the same order of magnitude when the sparse trellises are close to saturate.