Cluster Trellis: Data Structures & Algorithms for Exact Inference in Hierarchical Clustering

Craig S. Greenberg[*,1,2], **Sebastian Macaluso**[*,3], **Nicholas Monath**[1], **Ji Ah Lee**[1], **Patrick Flaherty**[1], **Kyle Cranmer**[3], **Andrew McGregor**[1], **Andrew McCallum**[1]
[1] University of Massachusetts Amherst
[2]National Institute of Standards and Technology
[3]New York University

Abstract

Hierarchical clustering is a fundamental task often used to discover meaningful structures in data. Due to the combinatorial number of possible hierarchical clusterings, approximate algorithms are typically used for inference. In contrast to existing methods, we present novel dynamic-programming algorithms for *exact* inference in hierarchical clustering based on a novel trellis data structure, and we prove that we can exactly compute the partition function, maximum likelihood hierarchy, and marginal probabilities of sub-hierarchies and clusters. Our algorithms scale in time and space proportional to the powerset of N elements, which is super-exponentially more efficient than explicitly considering each of the $(2N-3)!!$ possible hierarchies. Also, for larger datasets where our exact algorithms become infeasible, we introduce an approximate algorithm based on a sparse trellis that outperforms greedy and beam search baselines.

1 Introduction

Hierarchical clustering is often used to discover meaningful structures, such as phylogenetic trees of organisms (Kraskov et al., 2005), taxonomies of concepts (Cimiano and Staab, 2005), subtypes of cancer (Sørlie et al., 2001), and jets in particle physics (Cacciari et al., 2008). Among the reasons that hierarchical clustering has been found to be broadly useful is that it forms a natural data representation of data generated by a Markov tree, i.e., a tree-shaped model where the state variables are dependent only on their parent or children.

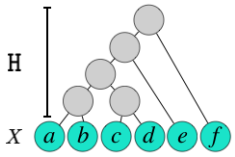

Figure 1: **Schematic representation of a hierarchical clustering**. H denotes the hierarchical clustering and X the dataset.

We define a hierarchical clustering as a recursive splitting of a dataset of N elements, $X = \{x_i\}_{i=1}^N$ into subsets until reaching singletons. This can equivalently be viewed as starting with the set of singletons and repeatedly taking the union of sets until reaching the entire dataset. We show a schematic representation in Figure 1, where we identify each x_i with a leaf of the tree and the hierarchical clustering as H. Formally,

Definition 1. (***Hierarchical Clustering***[1]) Given a dataset of elements, $X = \{x_i\}_{i=1}^N$, a **hierarchical clustering**, H, is a set of nested subsets of X, s.t. $X \in \mathtt{H}$, $\{\{x_i\}\}_{i=1}^N \subset \mathtt{H}$, and $\forall X_i, X_j \in \mathtt{H}$, either $X_i \subset X_j$, $X_j \subset X_i$, or $X_i \bigcap X_j = \emptyset$. Further, $\forall X_i \in \mathtt{H}$, if $\exists X_j \in \mathtt{H}$ s.t. $X_j \subset X_i$, then $\exists X_k \in \mathtt{H}$ s.t. $X_j \bigcup X_k = X_i$.

Given a subset $X_L \in \mathtt{H}$, then X_L is referred to as a cluster in H. When $X_P, X_L, X_R \in \mathtt{H}$ and $X_L \bigcup X_R = X_P$, we refer to X_L and X_R as children of X_P, and X_P the parent of X_L and X_R; if $X_L \subset X_P$ we refer to X_P as an ancestor of X_L and X_L a descendent of X_P.(We also denote the sibling of X_L, as $X_R = X_P \setminus X_L$.) For binary trees, the total number of possible pairs of siblings (X_L, X_R) for a parent with N elements is given by the Stirling number of the second kind $S(N, 2) = 2^{N-1} - 1$.

*The first two authors contributed equally.

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

[1]We limit our exposition to binary hierarchical clustering. Binary structures encode more tree-consistent clusterings than k-ary (Blundell et al., 2010). Natural extensions may exist for k-ary clustering, which are left for future work.

In our work, we consider an energy-based probabilistic model for hierarchical clustering. We provide a general (and flexible) definition of the probabilistic model and then give three specific examples of the distribution in section 4. Our model is based on measuring the compatibility of all pairs of sibling nodes in a binary tree structure. Formally,

Definition 2. (*Energy-based Hierarchical Clustering*) Let X be a dataset, $\mathtt{H}$ be a hierarchical clustering of X, let $\psi : 2^X \times 2^X \to \mathbb{R}^+$ be a potential function describing the compatibility of a pair of sibling nodes in $\mathtt{H}$, and let $\phi(X|\mathtt{H})$ be a potential function for the $\mathtt{H}$ structure. Then, the probability of $\mathtt{H}$ for the dataset X, $P(\mathtt{H}|X)$, is equal to the unnormalized potential of $\mathtt{H}$ normalized by the partition function, $Z(X)$:

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

where $\mathsf{sibs}(\mathtt{H}) = \{(X_L, X_R) | X_L \in \mathtt{H}, X_R \in \mathtt{H}, X_L \cap X_R = \emptyset, X_L \cup X_R \in \mathtt{H}\}$. The partition function $Z(X)$ is given by:

$$Z(X) = \sum_{\mathtt{H} \in \mathcal{H}(X)} \phi(X|\mathtt{H}). \tag{2}$$

where $\mathcal{H}(X)$ represents all binary hierarchical clusterings of the elements X.

We refer to our model as an energy-based model given that $\psi(\cdot, \cdot)$ is often defined by the unnormalized Gibbs distribution, i.e., $\psi(X_L, X_R) = \exp(-\beta E(X_L, X_R))$, where β is the inverse temperature and $E(\cdot, \cdot)$ is the energy. This probabilistic model allows us to express many familiar distributions over tree structures. It also has a connection to the classic algorithmic hierarchical clustering technique, agglomerative clustering, in that $\psi(\cdot, \cdot)$ has the same signature as a "linkage function" (i.e., single, average, complete linkage). We note that in this work we do not use informative prior distributions over trees $P(\mathtt{H})$ and instead assume a uniform prior.

Often, probabilistic approaches, such as coalescent models (Teh et al., 2008; Boyles and Welling, 2012; Hu et al., 2013) and diffusion trees (Neal, 2003; Knowles and Ghahramani, 2011), model which tree structures are likely for a given dataset. For instance, in particle physics generative models of trees are used to model jets (Cacciari et al., 2008), and similarly coalescent models have been used in phylogenetics (Suchard et al., 2018). Inference in these approaches is done by approximate, rather than exact, methods that lead to local optima, such as greedy best-first, beam-search, sequential Monte Carlo (Wang et al., 2015), and MCMC (Neal, 2003). Also, these methods do not have efficient ways to compute an exact normalized distribution over all tree structures.

Exactly performing MAP inference and finding the partition function by enumerating all hierarchical clusterings over N elements is exceptionally difficult because the number of hierarchies grows extremely rapidly, namely $(2N-3)!!$ (see (Callan, 2009; Dale and Moon, 1993) for more details and proof), where !! is double factorial. To overcome the computational burden, in this paper we introduce a cluster trellis data structure for hierarchical clustering. The cluster trellis, inspired by (Greenberg et al., 2018), enables us to use dynamic programming algorithms to exactly compute MAP structures and the partition function, as well as compute marginal distributions, including the probability of any sub-hierarchy or cluster. We further show how to sample exactly from the posterior distribution over hierarchical clusterings (i.e., the probability of sampling a given hierarchy is equal to the probability of that hierarchy). Our algorithms compute these quantities without having to iterate over each possible hierarchy in the $\mathcal{O}(3^N)$ time, which is super-exponentially more efficient than explicitly considering each of the $(2N-3)!!$ possible hierarchies (see Corollary 2 for more details). Thus, while still exponential, this is feasible in regimes where enumerating all possible trees would be infeasible, and is to our knowledge the fastest exact MAP/partition function result(See §A.5 and §A.7 for proofs), making practical exact inference for datasets on the order of 20 points ($\sim 3 \times 10^9$ operations vs $\sim 10^{22}$ trees) or fewer. For larger datasets, we introduce an approximate algorithm based on a sparse hierarchical cluster trellis and we outline different strategies for building this sparse trellis. We demonstrate our methods' capabilities for exact inference in discovering cascades of particle decays in jet physics and subtype hierarchies in cancer genomics, two applications where there is a need for exact inference on datasets made feasible by our methods. We find that greedy and beam search methods frequently return estimates that are sub-optimal compared to the exact MAP clustering.

Contributions of this Paper. We achieve *exact*, not approximate, solutions to the following:

- **Compute the Partition Function** $Z(X)$ (§2.2).
- **MAP Inference**, i.e. find the maximum likelihood tree structure $\text{argmax}_{\mathtt{H} \in \mathcal{H}} P(\mathtt{H}|X)$ (§2.3).
- **Sample Hierarchies from the Posterior Distribution**, i.e. weighted by their probability, $P(\mathtt{H}|X)$ (§2.5).

2 Hierarchical Cluster Trellis

Exactly performing MAP inference and finding the partition function by enumerating all hierarchical clusterings over N elements is intractable since the number of hierarchies grows extremely rapidly, namely $(2N-3)!!$ (see (Callan, 2009; Dale and Moon, 1993) for more details and proof), where !! is double factorial. To address this challenge, we introduce a cluster trellis data structure for hierarchical clustering. We describe how this

data structure enables us to use dynamic programming algorithms to exactly compute the partition function, MAP hierarchical clusterings, and marginals, as well as how to sample from the exact distribution over hierarchical clusterings.

2.1 Trellis Data Structure

The trellis data structure is a directed acyclic graph that encodes a set of hierarchical clusterings. Each vertex in the trellis corresponds to a node in a hierarchical clustering, and edges between vertices in the trellis correspond to a parent/child relationship in a hierarchical clustering. As in a hierarchical clustering, the trellis has a root node, that corresponds to the entire dataset, and leaf nodes that correspond to the individual elements of the dataset. The dataset associated with a trellis vertex $\mathbb{V}$ is denoted $X(\mathbb{V})$ and the trellis vertex associated with a dataset X is denoted $\mathbb{V}(X)$. Each vertex in the trellis stores memoized values of $Z(\mathbb{V})$ for computing the partition function, as well as the value $\phi(\mathtt{H}^*[\mathbb{V}])$ and the backpointer $\Xi(\mathtt{H}^*[\mathbb{V}])$ for computing the MAP tree. We denote $\mathbb{C}(X)$ as the children of $\mathbb{V}(X)$. We refer to a full trellis as the data structure where every possible hierarchical clustering given a dataset X can be realised, i.e., there is a bijection between the set of trellis vertices and $\mathbb{P}(X)\backslash\varnothing$, where $\mathbb{P}$ indicates the power set, and there is an edge between $\mathbb{V}_i$ and $\mathbb{V}_j$ if $X(\mathbb{V}_i) \subset X(\mathbb{V}_j)$. In contrast, a sparse trellis will only contain a subset of all possible hierarchies by omitting some of the vertices and edges in a full trellis.

2.2 Computing the Partition Function

Given a dataset of elements, $X = \{x_i\}_{i=1}^N$, the partition function, $Z(X)$, for the set of hierarchical clusterings over X, $\mathcal{H}(X)$, is given by Equation 2. The trellis implements a memoized dynamic program to compute the partition function and the MAP. To achieve this, we need to re-write the partition function in the corresponding recursive way. In particular,

Proposition 1. For any $x \in X$, the hierarchical partition function can be written recursively, as $Z(X) = \sum_{\mathtt{H}\in\mathcal{H}(X)} \phi(X|\mathtt{H}) = \sum_{X_i\in\mathbb{C}(X)_x} \psi(X_i, X\setminus X_i)\cdot Z(X_i)\cdot Z(X\setminus X_i)$ where $\mathbb{C}(X)_x$ is the set of all children of X containing the element x, i.e,. $\mathbb{C}(X)_x = \{X_j : X_j \in \mathbb{C}(X) \wedge x \in X_j\}$. In the particular case of a full trellis, then $\mathbb{C}(X)_x = \{X_j : X_j \in 2^X \setminus X \wedge x \in X_j\}$.

The proof is given in § A.1 in the Appendix. Algorithm 1 describes in a recursive way how to efficiently compute the partition function using the trellis based on Proposition 1. We first set the partition function of the leaf nodes in the trellis to 1. Then, we start by selecting any element in the dataset, x_i, and consider all clusters $X_i \in \mathbb{C}(X)$ such that $x_i \in X_i$. Next, the partition function is computed (memoized, recursively) for X_i and its complement $X \setminus X_i$, thus enabling the application of Proposition 1 to get $Z(X)$. For a full trellis, the algorithm can straightforwardly be written in a bottom-up, non-recursive way. In this case, the partition function for every node in the trellis is computed in order (in a bottom-up approach), from the nodes with the smallest number of elements to the nodes with the largest number of elements, memoizing the partition function value at each node. By computing the partial partition functions in this order, whenever computing the partition function of a given node in the trellis, the corresponding ones of all of the descendent nodes will have already been computed and memoized. In Figure 2, we show a visualization comparing the computation of the partition function with the trellis to the brute force method for a dataset of four elements. Next, we present the complexity result for Algorithm 1:

Algorithm 1 `PartitionFunction`(X)

```
Pick x_i ∈ X and set Z(X) ← 0
for X_i in C(X)_{x_i} do
  if Z(X_i) not set then
  Z(X_i) ← PartitionFunction(X_i)
  if Z(X \ X_i) not set then
  Z(X \ X_i) ← PartitionFunction(X \ X_i)
  Z(X) ← Z(X) + ψ(X_i, X \ X_i) · Z(X_i) · Z(X \ X_i)
return Z(X)
```

Theorem 1. For a given dataset X of N elements, Algorithm 1 computes $Z(X)$ in $\mathcal{O}(3^N)$ time.

The time-complexity of the algorithm is $\mathcal{O}(3^N)$, which is is significantly smaller than the $(2N-3)!!$ possible hierarchies.

Corollary 2. For a given dataset X of N elements, Algorithm 1 is super-exponentially more efficient than brute force methods that consider every possible hierarchy. In particular the ratio is $\mathcal{O}((\frac{2}{3})^N\ \Gamma(N-1/2))$.

The proofs of Algorithm 1 and Corollary 2 are given in § A.7 of the Appendix.

2.3 Computing the MAP Hierarchical Clustering

Similar to other dynamic programming algorithms, such as Viterbi, we can adapt Algorithm 1 in order to find the MAP hierarchical clustering.

The MAP clustering for dataset X, is $\mathtt{H}^\star(X) = \text{argmax}_{\mathtt{H}\in\mathcal{H}(X)}\ \phi(\mathtt{H})$. Here we can also use a recursive memoized technique, where each node will store a value for the MAP, denoted by $\phi(\mathtt{H}^\star(X))$ and a backpointer $\Xi(\mathtt{H}^\star(X))$. Specifically,

Proposition 2. For any $x \in \mathbb{C}(X)$, let $\mathbb{C}(X)_x = \{X_j : X_j \in \mathbb{C}(X) \wedge x \in X_j\}$, then $\phi(\mathtt{H}^\star(X)) = \max_{X_i\in\mathbb{C}(X)_x} \psi(X_i, X\setminus X_i)\cdot\phi(\mathtt{H}^\star(X_i))\cdot\phi(\mathtt{H}^\star(X\setminus X_i))$.

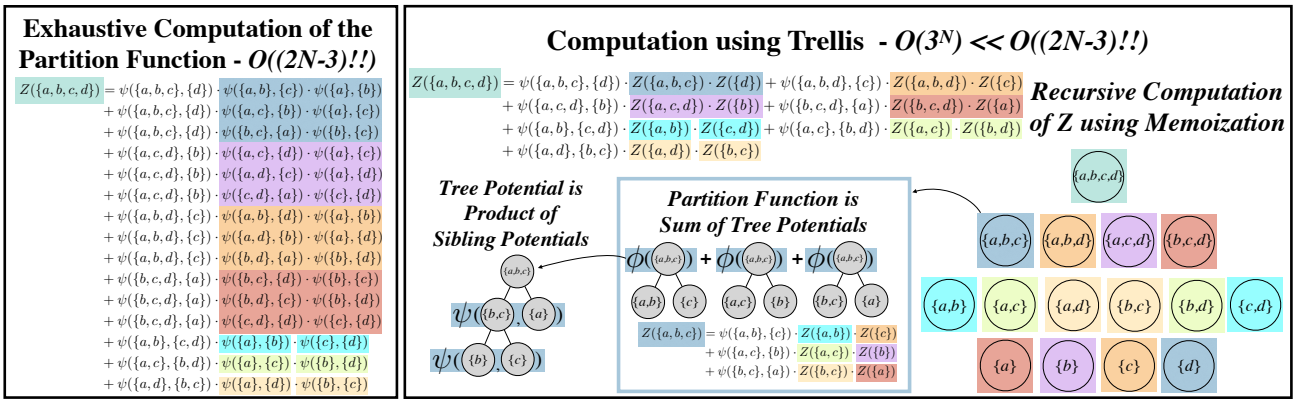

Figure 2: **Computing the partition function for the dataset** $\{a, b, c, d\}$. Left: exhaustive computation, consisting of the summation of $(2 \cdot 4 - 3)!! = 15$ energy equations. Right: computation using the trellis. The sum for the partition function is over $2^{4-1} - 1 = 7$ equations, each making use of a memoized Z value. Colors indicate corresponding computations over siblings in the trellis.

Algorithm 2 $\texttt{MAP}(X)$

if $\phi(X)$ set **then**
  return $\phi(X), \Xi(X)$
Pick $x_i \in X$
$\phi(X) \leftarrow -\infty$
$\Xi(X) \leftarrow \texttt{null}$ {Backpointer to give MAP tree structure.}
for X_i in $\mathbb{C}(X)_{x_i}$ **do**
  $t \leftarrow \psi(X_i, X \setminus X_i) \cdot \phi(\mathbb{V}(X_i)) \cdot \phi(\mathbb{V}(X \setminus X_i))$
  if $\phi(X) < t$ **then**
    $\phi(X) \leftarrow t$
    $\Xi(X) \leftarrow \{X_i, X \setminus X_i\} \cup \Xi(X_i) \cup \Xi(X \setminus X_i)$
return $\phi(X), \Xi(X)$

See §A.6 in the Appendix for the proof. As in the partition function algorithm described in Section 2.2, the time complexity for finding the MAP clustering is also $\mathcal{O}(3^N)$. The main difference is that to compute the maximal likelihood hierarchical clustering, the maximal energy of the sub-hierarchy rooted at each node is computed, instead of the partition function. Pointers to the children of the maximal sub-hierarchy rooted at each node are stored at that node. A proof of the time complexity, analogous to the one for the partition function, can be found in §A.5 of the Appendix.

2.4 Computing Marginals

In this section, we describe how to compute two types of marginal probabilities. The first is for a given sub-hierarchy rooted at X_i, i.e., $\texttt{H}_i \in \mathcal{H}(X_i)$, defined as $P(\texttt{H}_i|X) = \sum_{\texttt{H} \in A(\texttt{H}_i)} P(\texttt{H}|X)$, where $A(\texttt{H}_i) = \{\texttt{H} : \texttt{H} \in \mathcal{H}(X) \wedge \texttt{H}_i \subset \texttt{H}\}$, and $\texttt{H}_i \subset \texttt{H}$ indicates that $\texttt{H}_i$ is a subtree of $\texttt{H}$. Thus, we marginalize over every possible hierarchy while keeping fixed the sub-hierarchy $\texttt{H}_i$. The second is for a given cluster, X_i, defined as $P(X_i|X) = \sum_{\texttt{H} \in A(X_i)} P(\texttt{H}|X)$, where $A(X_i) = \{\texttt{H} : \texttt{H} \in \mathcal{H}(X) \wedge X_i \subset \texttt{H}\}$, and $X_i \subset \texttt{H}$ indicates that cluster X_i is contained in $\texttt{H}$. In this case, we marginalize over every possible sub-hierarchy that contains the cluster X_i while keeping the rest of the hierarchy $\texttt{H}$ fixed. The value of $P(\texttt{H}_i|X)$ can be computed using the same algorithm used for the partition function, except that we first merge $\texttt{H}_i$ into a single leaf node and use $\phi(\texttt{H}_i(X_i))$ for the energy of the newly merged leaf. The same is true for computing the value of $P(X_i|X)$, except that after merging X_i into a single leaf node, the value $Z(X_i)$ should be used. See Appendix § A.4 for proofs.

2.5 Sampling from the Posterior Distribution

Drawing samples from the true posterior distribution $P(\texttt{H}|X)$ is also difficult because of the extremely large number of trees. In this section, we introduce a sampling procedure for hierarchical clusterings $\texttt{H}_i$ implemented using the trellis which gives samples from the exact true posterior without enumerating all possible hierarchies.

The sampling procedure will build a tree structure in a top-down way. We start with the cluster of all the elements, X, then sample one child of that cluster, $X_L \subset X$, (Eq. 3) and set the other one to be the complement of X_L, i.e., $X \setminus X_L$. This is repeated recursively from each of the children and terminates when a cluster contains a single element. A child X_L of parent X_p, i.e., $X_L \subset X_p$ is sampled according to:

$$p(X_L|X_p) = \frac{1}{Z(X_p)} \cdot \psi(X_L, X_p \backslash X_L) \cdot Z(X_L) \cdot Z(X_p \backslash X_L). \tag{3}$$

Pseudocode for this algorithm is given in Algorithm 3.

Theorem 3. *Sample*(X) (Alg. 3) gives samples from $P(\texttt{H}|X)$.

The proof is given in Appendix § A.2. This algorithm is notable in that it does not require computing a categorical distribution over all trees and samples exactly according to $P(\texttt{H}|X)$.

Algorithm 3 `Sample`(X)

if $|X| = 1$ **return** $\{X\}$
Sample X_L from $p(X_i|X)$ (Eq. 3).
return $\{X_L,\ X \setminus X_L\} \cup$ `Sample`$(X_L) \cup$ `Sample`$(X \setminus X_L)$

3 Sparse Hierarchical Cluster Trellis

In this section, we introduce a sparse trellis data structure, which allows to scale to larger datasets by controlling the sparsity index, i.e. the fraction of hierarchies we consider from the total of $(2N-3)!!$. Most hierarchies have potential values orders of magnitude smaller than the MAP clustering making their contribution to the partition function negligible. As a result, if we build a sparse trellis that considers the most relevant hierarchies, we could find approximate solutions for inference in datasets where implementing the full trellis is not feasible. Conceptually, the only difference with respect to the full trellis is that the children of each vertex are typically a subset of all 2^X possible ones. Thus, the algorithms and proofs are the same as the ones presented in Section 2 but the solutions will be approximate. The specific vertices that are contained in the sparse trellis depend on how we build it. Below we present two possible strategies.

3.1 Building Strategies

The performance of the sparse trellis depends on the subset of all possible hierarchies over which it expands. This subset is chosen by the building strategy, which provides a sample of trees used to create the trellis. There are also different mappings for the ordering of the leaves of the input trees, and it is interesting to study the different subsets of hierarchies spanned by the sparse trellis depending on this mapping.

We start with a set of input trees. Once we choose a specific ordering of the leaves, we iterate over each input tree, creating a vertex $\mathbb{V}_i$ in the trellis for each new node in the tree, i.e. nodes that have not been visited in previous input trees. A schematic representation is shown in Figure 3. This way, the input sample of trees determines the trellis vertices that are created. The trellis considers every possible hierarchical clustering that can be realized with these vertices which is typically much greater than the number of input trees. After creating the trellis, we initialize the leaf vertices values with some dataset of interest and run the inference algorithms, e.g. MAP and partition function computations.

We emphasize that the approximate methods work precisely the same as in the exact method, and that the only difference is the exact algorithms use a full trellis, while the approximate algorithms use a sparse trellis. This means that the approximate algorithms find the optimal hierarchical clustering among those

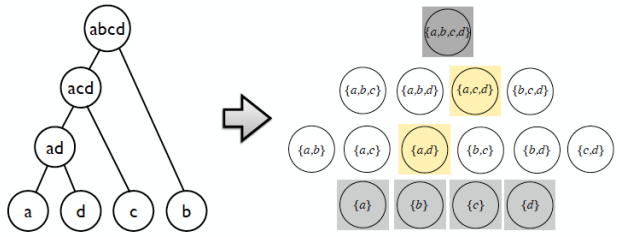

Figure 3: **Schematic representation of how the sparse trellis is built** iterating over each tree with four leaves from a sample dataset X. After every hierarchical structure is added, the final trellis is composed of the colored vertices, the added edges, the leaves and the root vertex. The vertices that are not colored represent the subset of vertices of the full trellis that are missing in the sparse case.

encoded by the sparse trellis, and thus the quality of the approximate hierarchical clustering is entirely dependent on the quality of hierarchical clusterings encoded by the sparse trellis.

Next, we present two distinctive procedures to build the trellis, which we refer to as Simulator trellis and Beam Search trellis. In both cases, a number of trees are generated, and then the union of the nodes and edges of these generated trees become the vertices and edges of the sparse trellis.

Simulator Trellis: in some cases there exists a generative model or simulator that implicitly defines a distribution over hierarchies. In the simulator trellis, we use this model/simulator to sample a set of trees that are used to seed the sparse trellis. We restrict the generated trees to have the same number of leaves, which is fixed for each trellis we create.

Beam Search Trellis: trees used to seed the sparse trellis are obtained by repeatedly running the beam search algorithm over a sample of sets of leaves. This approach is much more general than a simulator trellis, as it could be implemented for datasets where there is no generative model. Note that we choose beam search for our experiments, but this approach could be implemented with any agglomerative clustering, and only requires a "linkage function" (i.e., single, average, complete linkage).

4 Experiments

In this section, we demonstrate the use of the exact MAP, partition function, and sampling approaches described in this paper on two real world applications: jet physics and cancer genomics, as well as one synthetic data experiment related to Dasgupta's cost (Dasgupta, 2016). First, we give an illustrative example for the use of the proposed approaches with Dasgupta's cost, running on the kinds of data for which greedy methods are known to be approximate. In each real world application, we demonstrate how the trellis is used to compute exact MAP and the distribution over clusterings that are more informative and accurate than approximate

methods. In particle physics, we additionally demonstrate the use of the sampling procedure (§2.5) and the implementation of a sparse trellis. In cancer genomics, we show how we can model subtypes of cancer, which can help determine prognosis and treatment plans.

4.1 Dasgupta's Cost

Probabilistic model Dasgupta (2016) defines a cost function for hierarchical clustering that has been the subject of much theoretical interest (primarily on approximation algorithms for the cost) (Cohen-Addad et al., 2017, 2019; Charikar and Chatziafratis, 2017; Charikar et al., 2019; Moseley and Wang, 2017; Roy and Pokutta, 2017). Given a graph with vertices of the dataset X and weighted edges representing pairwise similarities between points $\mathcal{W} = \{(i, j, w_{ij})|i, j \in \{1, ..., |X|\} \times \{1, ..., |X|\}, i < j, w_{ij} \in \mathbb{R}^+\}$. Dasgupta's cost is defined as:

$$E(X_i, X_j) = (|X_i| + |X_j|) \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \quad (4)$$

This is equivalent to the cut-cost definition of Dasgupta's cost with the restriction to binary trees (Dasgupta, 2016).

Results Figure 4 gives an example graph, as proposed by (Charikar et al., 2019) to bound average-linkage performance, following a model for which greedy methods are known to be approximate with respect to Dasgupta's cost (Moseley and Wang, 2017; Cohen-Addad et al., 2017). We run greedy agglomerative clustering and trellis-based MAP procedure (Eq. 4). Unsurprisingly, the greedy method fails to achieve the lowest cost tree while the trellis-based method identifies an optimal tree. The cost of the greedily built tree is 44.08 while the tree built using the trellis is 40.08.

4.2 Jet Physics

Background The Large Hadron Collider (LHC) at CERN collides two beams of high-energy protons and produces many new (unstable) particles. Some of these new particles (quarks and gluons) will undergo a *showering process*, where they radiate many other quarks and gluons in successive binary splittings. These $1 \to 2$ splittings can be represented with a binary tree, where the energy of the particles decreases after each step. When the energy is below a given threshold, the showering terminates, resulting in a spray of particles that is called a *jet*. The particle detectors only observe the leaves of this binary tree (the jet constituents), and the unstable particles in the showering process are unobserved. Thus, a specific jet could result from several latent trees[2] generated by the showering process. While the latent showering process is unobserved, it is described by quantum chromodynamics (QCD).

Probabilistic Model The potential of a hierarchy is identified with the product of the likelihoods of all the $1 \to 2$ splittings of a parent cluster into two child clusters in the binary tree. Each cluster, X, corresponds to a particle with an energy-momentum vector $x = (E \in \mathbb{R}^+, \vec{p} \in \mathbb{R}^3)$ and squared mass $t(x) = E^2 - |\vec{p}|^2$. A parent's energy-momentum vector is obtained from adding its children, i.e., $x_P = x_L + x_R$. We study a toy model for jet physics (Cranmer et al., 2019a), where for each pair of parent and left (right) child cluster with masses $\sqrt{t_P}$ and $\sqrt{t_L}$ ($\sqrt{t_R}$) respectively, the likelihood function is,

$$\psi(X_L, X_R) = f(t(x_L)|t_P, \lambda) \cdot f(t(x_R)|t_P, \lambda) \quad (5)$$

$$\text{with} \quad f(t|t_P, \lambda) = \frac{1}{1 - e^{-\lambda}} \frac{\lambda}{t_P} e^{-\lambda \frac{t}{t_P}} \quad (6)$$

where the first term in $f(t|t_P, \lambda)$ is a normalization factor associated to the constraint that $t < t_P$.

Data and Methods We will compare full and sparse trellises results for the MAP hierarchical clustering with approximate methods, as described below. The ground truth hierarchical clusterings of our dataset are generated with the toy generative model for jets Ginkgo, see (Cranmer et al., 2019a) for more details. This model implements a recursive algorithm to generate a binary tree, whose leaves are the jet constituents. Jet constituents (leaves) and intermediate state particles (inner nodes) in Ginkgo are represented by a four dimensional energy-momentum vector.

Next, we review new implementations of greedy and beam search algorithms to cluster jets based on the joint likelihood of the jet binary tree in Ginkgo. The goal is to obtain the maximum likelihood estimate (MLE) or MAP for the latent structure of a jet. In this approach, the tree latent structure `H` is fixed by the algorithm. Greedy simply chooses the pairing of nodes that locally maximizes the likelihood at each step, whereas beam search maximizes the likelihood of multiple steps before choosing the latent path. The current implementation only takes into account one more step ahead, with a beam size given by $\frac{N(N-1)}{2}$, with N the number of jet constituents to cluster. Also, when two or more clusterings had an identical likelihood value, only one of them was kept in the beam, to avoid counting multiple times the different orderings of the same clustering (see (Boyles and Welling, 2012) for details about the different orderings of the internal nodes of the tree). This approach significantly improved the performance of beam search in terms of finding the MAP tree.

Results In this section we show results for a jet physics dataset of 5000 Ginkgo (Cranmer et al., 2019b)

[2]We refer to the trees as "latent" since an instance of a showering process has a corresponding tree, however that tree is unobserved.

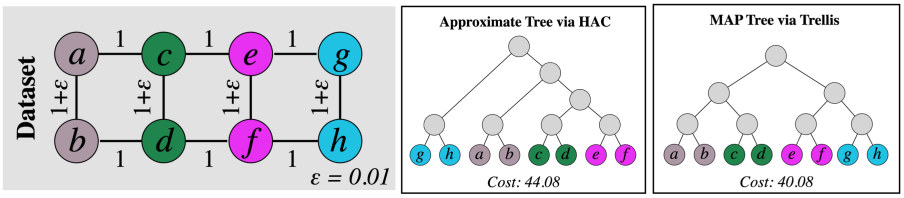

Figure 4: **Dasgupta's Cost.** Trellis vs agglomerative clustering MAP trees for a graph that is known to be difficult for with greedy methods.

Table 1: **Mean and standard deviation for the difference in log likelihood** for the MAP tree found by algorithms indicated by the row and column heading on the Ginkgo510 dataset.

	Beam Search	Greedy
Trellis	0.4 ± 0.5	1.5 ± 1.1
Beam Search		1.1 ± 1.1

jets with a number of leaves between 5 and 10, and we refer to it as Ginkgo510. We start by comparing in Table 1 the mean difference among the MAP values for the hierarchies log likelihood obtained with the full trellis, beam search and greedy algorithms. We see that the likelihood of the trees increases from greedy to beam search to the trellis one, as expected. Next, in Figure 5 we show the partition function versus the MAP hierarchy for each set of leaves in Ginkgo510 dataset. It is interesting to note that there seems to be a correlation between Z and the Trellis MAP.

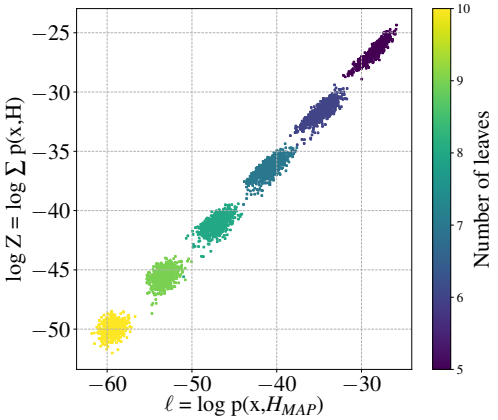

Figure 5: **Scatter plot of the partition function Z vs. the trellis MAP value** ℓ for Ginkgo510 dataset, with up to 10 leaves (jet constituents). The color indicates the number of leaves of each hierarchical clustering. There appears to be a correlation between Z and the MAP values.

Next, we show an implementation of the sampling procedure introduced in section 2.5. We compare in Figure 6 the results from sampling 10^5 hierarchies (black dots) and the expected distribution[3] (green) for the likelihood of each hierarchy. There is an excellent agreement between the sampled and the expected distributions. Here we showed, for illustrative purposes, a way to estimate the posterior distribution using our sampling procedure. However, we want to emphasize that the key contribution of our procedure is that it allows to sample hierarchies from the exact true posterior distribution, i.e. sample a hierarchy according to its probability.

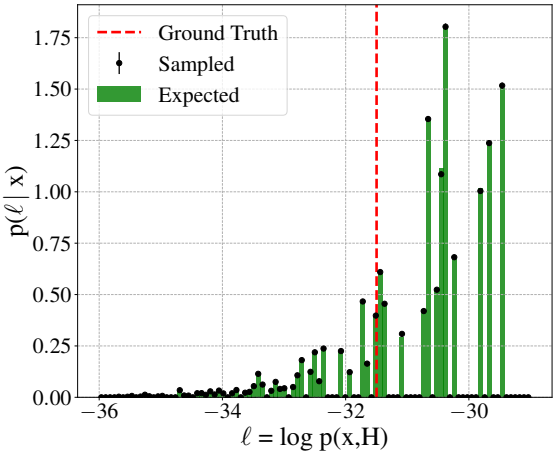

Figure 6: **Comparison of the posterior distribution** for a specific jet with five leaves for sampling 10^5 hierarchies using Alg 3 (black dots with small error bars) and expected posterior distribution (in green). The plots show the discrete nature of the distribution. The log likelihood for the ground truth tree is a vertical dashed red line.

Finally, as a proof of concept, we show in Figure 7 the performance of the sparse trellis to calculate the MAP values on a set of 100 Ginkgo jets with 9 leaves. This illustrates the relationship between the effectiveness and sparsity observed in our experiments, where a higher value on the y-axis represents greater effectiveness and a smaller value on the x-axis represents greater sparsity. We chose a dataset of 9 elements to be able to easily compare the performance of the sparse and full trellises. However, the sparse trellis can be applied to larger datasets. Even though beam search has a good performance for trees with a small number of leaves, we see that both sparse trellises quickly improve over beam search, with a sparsity index of only about 2%. Both sparse trellises approach the performance of the exact one, but the BS trellis does it sooner. Also, in

[3]The expected posterior is defined as the probability density function of each possible hierarchy. In principle, this could be obtained by taking the ratio of the likelihood of each hierarchy with respect to the partition function Z. We opt to take an approximate approach, as follows. If we sample enough number of times, we would expect each possible hierarchy to appear at least once. Thus, as a proof of concept, we sample 10^5 hierarchies for a set of five leaves (88 different hierarchies), keep only one of them for each unique likelihood value and normalize by Z and bin size. We show this result in the histogram labeled as Expected (green) in Figure 6.

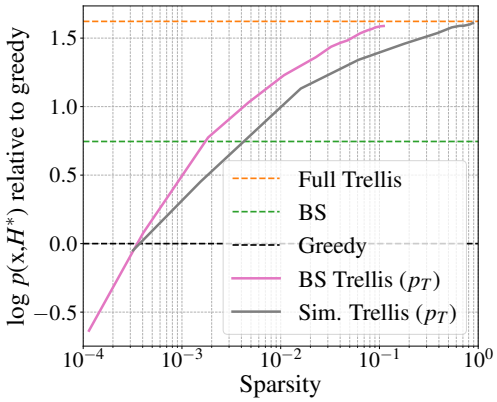

Figure 7: **Trellises MAP hierarchy log likelihood vs their sparsity**. MAP hierarchy log likelihood values are relative to the greedy algorithm. Each value corresponds to the mean over 100 trees of a test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises. In both cases, we present the trellis obtained by ordering the leaves of the input trees in increasing norm of their momentum vector $\vec{p} \in \mathbb{R}^3$ (see the probabilistic model description of section 4.2 for more details). We compare sparse trellises with other orderings of the leaves in Appendix § A.12. We add the values of the exact 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. Also, the sparse trellises are pre-built and then run on new datasets (test), which is why BS performs better than BS trellis sometimes.

Figure 13 in Appendix § A.12 we compare the empirical running times of the algorithms on the same dataset.

4.3 Cancer Genomics

Background Hierarchical clustering is a common clustering approach for gene expression data (Sørlie et al., 2001). It is not uncommon to have a need for clustering a small number of samples in cancer genomics studies. An analysis of data available from `https://clinicaltrials.gov` shows that the median sample size for 7,412 completed phase I clinical trials involving cancer is only 30.

Probabilistic Model In this case we are given a dataset of vectors indicating the level of gene expressions which are endowed with pairwise affinities that are both positive and negative. We define the energy of a pair of sibling nodes in the tree to be the sum of the across-cluster positive edges, minus the sum of negative within-cluster edge weights.

$$E(X_i, X_j) = \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \mathbb{I}[w_{ij} > 0] - \sum_{\substack{x_i, x_j \in X_i \times X_i, \\ x_i < x_j}} w_{ij} \mathbb{I}[w_{ij} < 0] - \sum_{\substack{x_i, x_j \in X_j \times X_j, \\ x_i < x_j}} w_{ij} \mathbb{I}[w_{ij} < 0] \tag{7}$$

where w_{ij} is the affinity between x_i and x_j. The correlation clustering input can be represented as a complete weighted graph, $G = (V, E)$, where each edge has weight $w_{uv} \in [-1, 1], \forall (u, v) \in E$. The goal is to construct a clustering of the nodes that maximizes the sum of positive within-cluster edge weights minus the sum of all negative across-cluster edge weights (since we wish to minimize the energy function given by Equation 7). This energy is the correlation clustering objective (Bansal et al., 2004).

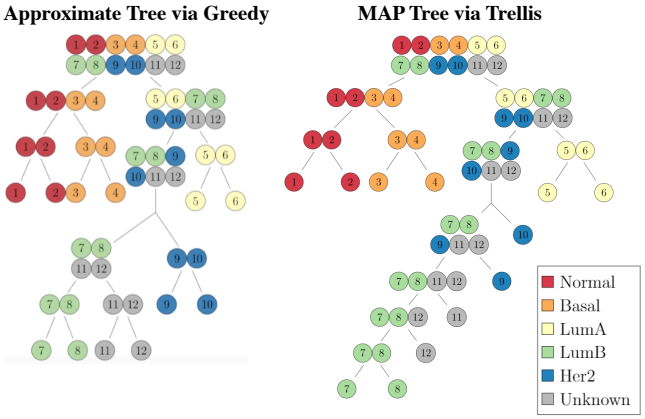

Figure 8: **Cancer Genomics**. Comparison of trees from greedy hierarchical clustering (left) and exact MAP clustering using the trellis (right) on the subsampled `pam50` data set. The colors indicate subtypes of breast cancer (grey if unknown). Though both appear to assign unknown samples to LumB, the right tree positions the unknown samples closer to the Her2 samples.

Data and Methods Here, we compare a greedy agglomerative clustering to our exact MAP clustering tree using the Prediction Analysis of Microarray 50 (`pam50`) gene expression data set. The `pam50` data set ($n = 232$, $d = 50$) is available from the UNC MicroArray Database (University of North Carolina, 2020). It has intrinsic subtype annotations for 139 of the 232 samples. Missing data values (2.65%) were filled in with zeros. We drew a stratified sample of the total data set with two samples from each known intrinsic subtype and two samples from the unknown group.

Results Figure 8 displays the greedy hierarchical clustering tree and the MAP tree with transformed weights for the twelve samples selected from the pam50 dataset. (The correlations among subsampled `pam50` ($n = 12$) data set are all positive.) The main difference between these trees is in the split of the subtree including LumB, HER2, and unknown samples. The greedy method splits HER2 from LumB and unknown, while the MAP tree shows a different topology for this subtree. For the MAP solution, we note that the subtree rooted at $\{7, 8, 9, 10, 11, 12\}$ is consistent. All of the correlation coefficients among this cluster are positive, so the optimal action is to split off the item with the smallest (positive) correlation coefficient.

4.4 Relationship Between Cost Functions

There are several measures of hierarchical clustering quality that are popular in the community. In addition to the Dasgupta cost and Hierarchical Correlation Clustering (HCC) objectives, which we discuss above, Dendrogram Purity (DP) is often used to measure the

quality of hierarchical clusterings when a ground truth flat clustering is available. We briefly discuss here how these three measures relate.

The degree to which Dasgupta cost and the HCC objectives correlate to DP is a function of how closely the pair wise edge weights reflect the ground truth clustering. To drive this point home, as an extreme, one could imagine adversarial edge weights, where the MAP hierarchical clusterings according to Dasgupta/HCC is un/negatively correlated with the hierarchical clusterings with maximal DP. In particular: (1) Maximal DP can be achieved by making a forest, where each tree consists solely of within cluster elements. Any tree that contains any such forest as subtrees is a maximal with respect to DP, and any such tree would have DP = 1. (2) Given 1/0 edge weights for within/across ground truth classes, respectively, the MAP Dasgupta cost could also be obtained by making a forest, where each tree consists solely of within cluster elements. Any tree that contains any such forest as subtrees is a MAP tree with respect to Dasgupta cost. In this case, the set of maximal DP trees and the set of MAP Dasgupta cost trees should be the same. (3) The same is true for HCC (but with edge weights set as +/- 1 for within/across ground truth classes). (4) If the edge weights are selected randomly, the MAP Dasgupta/HCC trees will be uncorrelated with DP. (5) If the edge weights are selected as -1 * edge weights described in (2) or (3) above, any MAP Dasgupta or HCC tree will achieve the worst possible cost with respect to DP.

5 Related Work

Modeling distributions over tree structures has been the subject of a large body of work. Bayesian nonparametric models typically define a posterior distribution over tree structures given data such as diffusion trees coalescents, and others (Neal, 2003; Teh et al., 2008, inter alia). These methods, while providing a distribution over trees, only support using parametric distributions to define emission probabilities rather than the energy-based model used in this paper. The Bayesian hierarchical clustering (BHC) model (Heller and Ghahramani, 2005b) is akin to the energy-based ones used in this paper. Inference includes greedy agglomerative (Heller and Ghahramani, 2005b), randomized (Heller and Ghahramani, 2005a), and tree re-arrangement approaches (Xu et al., 2009). Future work could consider how to use the trellis for BHC. Interestingly, the BHC likelihood is a mixture of tree consistent partitions, also related to using the trellis for flat clustering. Factor graph-based distributions over tree structures such as (Wick et al., 2012) on the other hand support a flexible class of distributions over tree structures as in our approach. However inference in factor graph models as well as many of the Bayesian non-parameteric models is typically approximate or performed by sampling methods. This lends in practice to approximate MAP solutions and distributions over tree structures. Exact methods like the one proposed in this paper have not, to our knowledge, been proposed.

Dasgupta (2016) defines a cost function for hierarchical clustering. Much work has been done to develop approximate solution methods and related objectives (Moseley and Wang, 2017, inter alia).

Bootstrapping methods, such as (Suzuki and Shimodaira, 2006), represent uncertainty in hierarchical clustering. Unlike our approach, bootstrapping methods approximate statistics of interest through repeatedly (re-)sampling from the empirical distribution.

Work on exact inference and exact distributions over flat clusterings (Greenberg et al., 2018), provides the foundation of our dynamic programming approach. Other work on exact flat clustering uses fast convolutions via the Mobius transform and Mobius inversion (Kohonen and Corander, 2016). Kappes et al. (2015) produce approximate distributions over flat clusterings using Perturb and MAP (Papandreou and Yuille, 2011).

Orthogonal to our work on uncertainty in hierarchical clustering, recent work has proposed continuous representations of trees for hierarchical clustering (Monath et al., 2019; Chami et al., 2020). This work represents uncertainty of child-parent assignments by considering the distance between two nodes in embedding space. We note that the distribution over trees used in these papers does not directly correspond to the energy-based distribution proposed in our work.

6 Conclusion

This paper describes a trellis data structure and dynamic-programming algorithm to efficiently compute and sample from probability distributions over hierarchical clusterings. Our method improves upon the computation cost of brute-force methods from $(2N-3)!!$ to sub-quadratic in the substantially smaller power-set of N, which is super-exponentially more efficient. We demonstrate our methods' utility on jet physics and cancer genomics datasets, as well as a dataset related to Dasgupta's cost (Dasgupta, 2016), and show its improvement over approximate methods. Also, for larger datasets where the full trellis implementation becomes infeasible, we introduce a sparse trellis that compares well to other benchmarks. Finally, our methods allow to sample hierarchies from the exact true posterior distribution without enumerating all possible ones, i.e. sample a hierarchy according to its probability. Code for our methods of finding exact solutions for the MAP hierarchy and partition function for any user-defined energy-based model of hierarchical clustering is available here: `https://github.com/SebastianMacaluso/ClusterTrellis`.

7 Acknowledgements

Kyle Cranmer and Sebastian Macaluso are supported by the National Science Foundation under the awards ACI-1450310 and OAC-1836650 and by the Moore-Sloan data science environment at NYU. Patrick Flaherty is supported in part by NSF HDR TRIPODS award 1934846. Andrew McCallum and Nicholas Monath are supported in part by the Center for Data Science and the Center for Intelligent Information Retrieval, and in part by the National Science Foundation under Grant No. NSF-1763618. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

References

N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine learning*, 2004.

C. Blundell, Y. Teh, and K. Heller. Bayesian rose trees. *Uncertainty in Artificial Intelligence, (UAI)*, 2010.

L. Boyles and M. Welling. The time-marginalized coalescent prior for hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

M. Cacciari, G. P. Salam, and G. Soyez. The anti-k_t jet clustering algorithm. *JHEP*, 2008.

D. Callan. A combinatorial survey of identities for the double factorial. *arXiv*, 2009.

I. Chami, A. Gu, V. Chatziafratis, and C. Ré. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

M. Charikar and V. Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017.

M. Charikar, V. Chatziafratis, and R. Niazadeh. Hierarchical clustering better than average-linkage. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2019.

P. Cimiano and S. Staab. Learning concept hierarchies from text with a guided agglomerative clustering algorithm. *Proceedings of the ICML 2005 Workshop on Learning and Extending Lexical Ontologies with Machine Learning Methods*, 2005.

V. Cohen-Addad, V. Kanade, and F. Mallmann-Trenn. Hierarchical clustering beyond the worst-case. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM)*, 2019.

K. Cranmer, S. Macaluso, and D. Pappadopulo. Toy Generative Model for Jets. *Toy Generative Model for Jets*, 2019a.

K. Cranmer, S. Macaluso, and D. Pappadopulo. Toy Generative Model for Jets Package. *https://github.com/SebastianMacaluso/ToyJetsShower*, 2019b.

E. Dale and J. Moon. The permuted analogues of three Catalan sets. *Journal of Statistical Planning and Inference*, 1993.

S. Dasgupta. A cost function for similarity-based hierarchical clustering. *Symposium on Theory of Computing (STOC)*, 2016.

C. Greenberg, N. Monath, A. Kobren, P. Flaherty, A. McGregor, and A. McCallum. Compact representation of uncertainty in clustering. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

K. Heller and Z. Ghahramani. Randomized algorithms for fast bayesian hierarchical clustering. In *PASCAL Workshop on Statistics and Optimization of Clustering*, 2005a.

K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. *Proceedings of the 22nd international conference on Machine learning*, 2005b.

Y. Hu, J. L. Ying, H. Daume III, and Z. I. Ying. Binary to bushy: Bayesian hierarchical clustering with the beta coalescent. *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

J. H. Kappes, P. Swoboda, B. Savchynskyy, T. Hazan, and C. Schnörr. Probabilistic correlation clustering and image partitioning using perturbed multicuts. *International Conference on Scale Space and Variational Methods in Computer Vision*, 2015.

D. A. Knowles and Z. Ghahramani. Pitman-yor diffusion trees. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.

J. Kohonen and J. Corander. Computing exact clustering posteriors with subset convolution. *Communications in Statistics-Theory and Methods*, 2016.

A. Kraskov, H. Stögbauer, R. G. Andrzejak, and P. Grassberger. Hierarchical clustering using mutual information. *EPL (Europhysics Letters)*, 2005.

G. Louppe, K. Cho, C. Becot, and K. Cranmer. QCD-Aware Recursive Neural Networks for Jet Physics. *JHEP*, 2019.

N. Monath, M. Zaheer, D. Silva, A. McCallum, and A. Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. *Conference on Knowledge Discovery & Data Mining (KDD)*, 2019.

B. Moseley and J. Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

R. M. Neal. Density modeling and clustering using dirichlet diffusion trees. *Bayesian statistics*, 2003.

G. Papandreou and A. L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. *2011 International Conference on Computer Vision*, 2011.

A. Roy and S. Pokutta. Hierarchical clustering via spreading metrics. *The Journal of Machine Learning Research*, 2017.

T. Sørlie, C. M. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. B. Eisen, M. Van De Rijn, S. S. Jeffrey, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences*, 2001.

M. A. Suchard, P. Lemey, G. Baele, D. L. Ayres, A. J. Drummond, and A. Rambaut. Bayesian phylogenetic and phylodynamic data integration using beast 1.10. *Virus evolution*, 2018.

R. Suzuki and H. Shimodaira. Pvclust: an r package for assessing the uncertainty in hierarchical clustering. *Bioinformatics*, 2006.

Y. W. Teh, H. Daume III, and D. M. Roy. Bayesian agglomerative clustering with coalescents. *Advances in Neural Information Processing Systems (NeurIPS)*, 2008.

University of North Carolina. UNC microarray database. *https://genome.unc.edu/*, 2020.

L. Wang, A. Bouchard-Côté, and A. Doucet. Bayesian phylogenetic inference using a combinatorial sequential monte carlo method. *Journal of the American Statistical Association*, 2015.

M. Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. *Association for Computational Linguistics (ACL)*, 2012.

Y. Xu, K. Heller, and Z. Ghahramani. Tree-based inference for dirichlet process mixtures. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

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 $(2N - 3)!!$

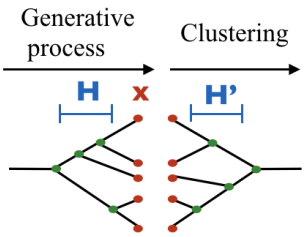

Figure 9: **Schematic representation of the tree structure of a sample jet** generated with Ginkgo and the clustered tree for some clustering algorithm. For a given algorithm, z labels the different variables that determine the latent structure of the tree. The tree leaves x are labeled in red and the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

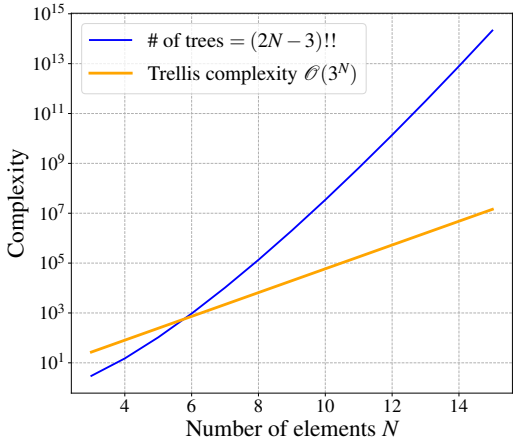

Figure 10: **Comparison of the complexity** of the cluster trellis (orange) and the number of trees (blue) vs the number of elements of a dataset.

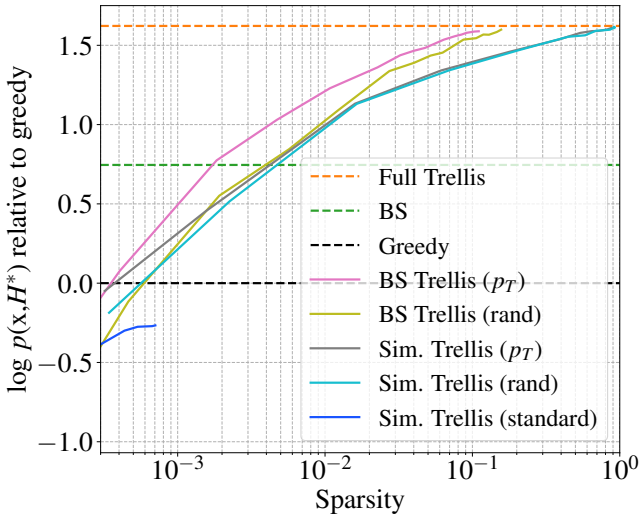

Figure 11: **Trellises MAP hierarchy log likelihood vs their sparsity**. MAP hierarchy log likelihood values are relative to the greedy algorithm. Each value corresponds to the mean over 100 trees of a test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises. We present the trellis 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.

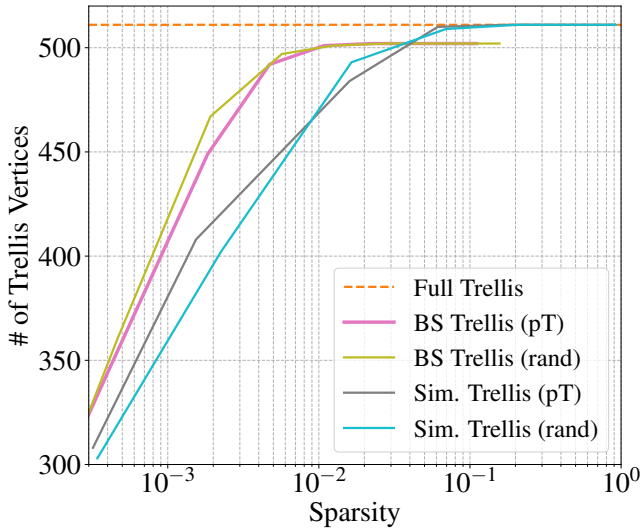

Figure 12: **Trellises number of vertices vs their sparsity**. We show the Simulator (Sim.) and the Beam Search (BS) trellises. The leaves of the input trees are ordered in different ways, as explained in Figure 11. Sim. trellis saturates all the vertices below a sparsity of ~ 0.1 but the performance 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).

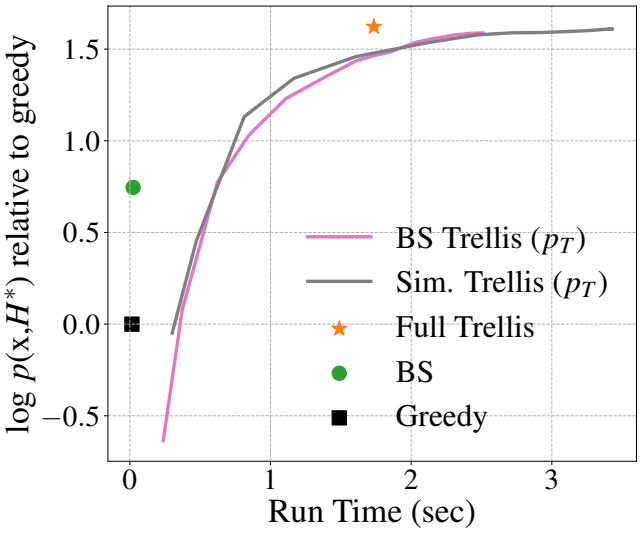

Figure 13: **MAP hierarchy log likelihood vs algorithms running time** on a set of 100 Ginkgo jets with 9 leaves. Each value corresponds to the mean over 100 trees of a test dataset. The difference between the sparse and exact trellises running times is because the exact one is iterative over the nodes 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.