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

Craig S. Greenberg*1,2, Sebastian Macaluso*3, Nicholas Monath1, Ji Ah Lee1, Patrick Flaherty1, Kyle Cranmer3, Andrew McGregor1, Andrew McCallum1

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. $X \in H$, $\{\{x_i\}\}_{i=1}^N \subset H$, and $\forall X_i, X_j \in H$, either $X_i \subset X_j$, $X_j \subset X_i$, or $X_i \cap X_j = \emptyset$. Further, $\forall X_i \in H$, if $\exists X_j \in H$ s.t. $X_j \subset X_i$, then $\exists X_k \in H$ s.t. $X_j \cup X_k = X_i$.

Given a subset $X_L \in H$, then $X_L$ is referred to as a cluster in $H$. When $X_P, X_L, X_R \in H$ and $X_L \cup X_R = X_P$, 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 $X_R$ 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.

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, $\mathcal{H}$ 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 $\mathcal{H}$, and let $\phi(X|\mathcal{H})$ be a potential function for the $\mathcal{H}$ structure. Then, the probability of $\mathcal{H}$ for the dataset $X$, $P(\mathcal{H}|X)$, is equal to the unnormalized potential of $\mathcal{H}$ normalized by the partition function, $Z(X)$:

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

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

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

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 $\beta$ 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 $P(\mathcal{H})$ 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 $O(3^N)$ 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** $Z(X)$ ($\S$2.2).
- **MAP Inference**, i.e. find the maximum likelihood tree structure $\arg \max_{\mathcal{H} \in \mathcal{H}} P(\mathcal{H}|X)$ ($\S$2.3).
- **Sample Hierarchies from the Posterior Distribution**, i.e. weighted by their probability, $P(\mathcal{H}|X)$ ($\S$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 $V$ is denoted $X(V)$ and the trellis vertex associated with a dataset $X$ is denoted $V(X)$. Each vertex in the trellis stores memoized values of $Z(V)$ for computing the partition function, as well as the value $\phi(H^*[V])$ and the backpointer $\Xi(H^*[V])$ for computing the MAP tree. We denote $C(X)$ as the children of $X$. 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 $\mathbb{P}(X)\setminus\emptyset$, where $\mathbb{P}$ indicates the power set, and there is an edge between $V_i$ and $V_j$ if $X(V_i) \subset X(V_j)$. 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 $Z(X) = \sum_{\mathcal{H} \in \mathcal{H}(X)} \phi(X | H) = \sum_{X_i \in C(X)} \psi(X_i, X \setminus X_i) \cdot Z(X_i) \cdot Z(X \setminus X_i)$ where $\mathcal{C}(X) = \{\forall x \in X \setminus x \in X_j\}$ is the set of all children of $X$ containing the element $x$, i.e., $\mathcal{C}(X) = \{X_j : x \notin \mathcal{C}(X) \land x \in X_j\}$. In particular, for a full trellis, $\mathcal{C}(X) = \{X_j : x \notin \mathcal{C}(X) \land x \in X_j\}$.

The proof is given in $\S$ 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 $X_i \in \mathcal{C}(X)$ 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 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 $x_i \in X$ and set $Z(X) \leftarrow 0$

for $X_i \in \mathcal{C}(X)$ do

if $Z(X_i)$ not set then

$Z(X_i) \leftarrow \text{PartitionFunction}(X_i)$

if $Z(X \setminus X_i)$ not set then

$Z(X \setminus X_i) \leftarrow \text{PartitionFunction}(X \setminus X_i)$

$Z(X) \leftarrow Z(X) + \psi(X_i, X \setminus X_i) \cdot Z(X_i) \cdot Z(X \setminus X_i)$

return $Z(X)$

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

The time-complexity of the algorithm is $O(3^N)$, which 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 $O((\frac{2}{3})^N \Gamma(N - 1/2))$.

The proofs of Algorithm 1 and Corollary 2 are given in $\S$ 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 $H^*(X) = \arg\max_{H \in \mathcal{H}(X)} \phi(H)$. Here we can also use a recursive memoized technique, where each node will store a value for the MAP, denoted by $\phi(H^*(X))$ and a backpointer $\Xi(H^*(X))$. Specifically,

**Proposition 2.** For any $x \in \mathcal{C}(X)$, let $\mathcal{C}(X)_x = \{X_j : x \in \mathcal{C}(X) \land x \in X_j\}$, then $\phi(H^*(X)) = \max_{X_i \in \mathcal{C}(X)_x} \psi(x_i, X \setminus X_j) \cdot \phi(H^*(X_i)) \cdot \phi(H^*(X \setminus X_i))$. 

C. Greenberg, S. Macaluso, N. Monath, J. Lee, P. Flaherty, K. Cranmer, A. McGregor, A. McCallum
Algorithm 2 MAP(X)

if \( \phi(X) \) set then
return \( \phi(X), \Xi(X) \)

Pick \( x_i \in X \)

\( \phi(X) \leftarrow -\infty \)

\( \Xi(X) \leftarrow \text{null} \) {Backpointer to give MAP tree structure.}

for \( X_i \in \mathcal{C}(X) \), do

\( t \leftarrow \psi(X_i, X \setminus X_i) \cdot \phi(V(X_i)) \cdot \phi(V(X \setminus X_i)) \)

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 \( O(3^N) \). 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., \( H_i \in \mathcal{H}(X_i) \), defined as

\[
P(H_i|X) = \sum_{H \in \mathcal{A}(H_i)} P(H|X),
\]

where \( \mathcal{A}(H_i) = \{ H : H \in \mathcal{H}(X) \land H_i \subset H \} \), and \( H_i \subset H \) indicates that \( H_i \) is a subtree of \( H \). Thus, we marginalize over every possible hierarchy while keeping fixed the sub-hierarchy \( H_i \). The second is for a given cluster, \( X_i \), defined as

\[
P(X_i|X) = \sum_{H \in \mathcal{A}(H_i)} P(H|X),
\]

where \( \mathcal{A}(H_i) = \{ H : H \in \mathcal{H}(X) \land X_i \subset H \} \), and \( X_i \subset H \) indicates that cluster \( X_i \) is contained in \( H \). In this case, we marginalize over every possible sub-hierarchy that contains the cluster \( X_i \) while keeping the rest of the hierarchy \( H \) fixed. The value of \( P(H_i|X) \) can be computed using the same algorithm used for the partition function, except that we first merge \( H_i \) into a single leaf node and use \( \phi(H_i(X_i)) \) 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 \( P(H|X) \) is also difficult because of the extremely large number of trees. In this section, we introduce a sampling procedure for hierarchical clusterings \( H_i \) 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:

\[
p(X_L|X_p) = \frac{1}{Z(X_L)} \cdot \psi(X_L, X_p \setminus X_L) \cdot Z(X_L) \cdot Z(X_p \setminus X_L).
\]

(3)

Pseudocode for this algorithm is given in Algorithm 3.

Theorem 3. Sample \( X \) (Alg. 3) gives samples from \( P(H|X) \).

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 \( P(H|X) \).
We emphasize that the approximate methods work. We start with a set of input trees. Once we choose a
Algorithm 3. Sample(X)

<table>
<thead>
<tr>
<th>if</th>
<th>return</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{X}</td>
</tr>
<tr>
<td></td>
<td>Sample X_{L} from (p(X_{L}</td>
</tr>
<tr>
<td></td>
<td>return ({X_{L}, X \setminus X_{L} } \cup \text{Sample}(X_{L}) \cup \text{Sample}(X \setminus X_{L}))</td>
</tr>
</tbody>
</table>

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 \((2^N - 3)!!\). 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 \(V_i\) 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 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 $W = \{(i, j, w_{ij}) | i, j \in \{1, ..., |X|\} \times \{1, ..., |X|\}, i < j, w_{ij} \in \mathbb{R}^+\}$. 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 $1 \rightarrow 2$ 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 $1 \rightarrow 2$ 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,

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

with

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

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 $\mathbf{H}$ is fixed by the algorithm. Greedily 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)
we refer to it as Ginkgo510. We start by comparing
Table 1:
we sample enough number of times, we would expect each
of concept, we sample 10
possible hierarchy to appear at least once. Thus, as a proof
of each hierarchy with respect to the partition function
this could be obtained by taking the ratio of the likelihood
density function of each possible hierarchy. In principle,
and the expected distribution

$Z = \log p(x, H)$

Next, we show an implementation of the sampling pro-
iques for a set of 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.

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.

5 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


Figure 5: Scatter plot of the partition function $Z$ vs.
the trellis MAP value $\ell$ 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 Trellis MAP.

Next, we show an implementation of the sampling pro-
duced introduced in section 2.5. We compare in Figure
6 the results from sampling $10^5$ hierarchies (black dots)
and the expected distribution5 (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,
not sample a hierarchy according to its probability.

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 illus-
trusts 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 spar-
sity. 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


<table>
<thead>
<tr>
<th>Beam Search</th>
<th>Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trellis</td>
<td>0.4 ± 0.5</td>
</tr>
<tr>
<td>Beam Search</td>
<td>1.1 ± 1.1</td>
</tr>
</tbody>
</table>

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 val-
ues 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 4: Dasgupta’s Cost.
Trellis vs agglomerative cluster-
ing MAP trees for a graph that is known to be difficult
for with greedy methods.

$\ell = \log p(x, H)$

$Z = \log \sum p(x, H)$

$\ell = \log p(x, H_{MAP})$

$\ell = \log p(x, H)$

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.
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). However, standard hierarchical clustering uses a greedy agglomerative or divisive heuristic to build a tree. 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.

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

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).

#### 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 non-parametric 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 and do not support the general family of potential functions used in our approach, which can use any scoring function to define the distribution. 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-parametric 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


