April 2021

COMPACT REPRESENTATIONS OF UNCERTAINTY IN CLUSTERING

Craig Stuart Greenberg
University of Massachusetts Amherst

COMPACT REPRESENTATIONS OF UNCERTAINTY IN CLUSTERING

A Dissertation Presented

by

CRAIG GREENBERG

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2021

College of Information and Computer Sciences

COMPACT REPRESENTATIONS OF UNCERTAINTY IN CLUSTERING

A Dissertation Presented

by

CRAIG GREENBERG

Approved as to style and content by:

Andrew McCallum, Chair

Andrew McGregor, Member

Arya Mazumdar, Member

Patrick Flaherty, Member

Alexander Schwing, Member

James Allan, Chair of the Faculty
College of Information and Computer Sciences

DEDICATION

To my family, friends, communities, and all those with whom I did, do, or will share a bond of love, in this or any life.

ACKNOWLEDGMENTS

A few years back, when I was going through a box of my childhood things, I came across an assignment of mine from the second grade. It was a little booklet about me–my favorite color is red, my favorite dog is my dog, that sort of thing. I was surprised to read that my favorite subject was math, since my only math-related memories from around that time were from preschool, and those were of being bored by counting lessons. I recalled sleeping through my math classes in junior and high school, though still performing well on tests, largely avoiding math at Vanderbilt, where I studied percussion[1], and having a most rude awakening when I studied combinatorics[1] at Penn–I found the subject impossibly hard and barely passed the class, despite my great efforts! Considering this and the fact that a bulk of my Ph.D. thesis utilizes combinatorics underscores that it has been a long road to this point, and makes somehow more obvious that this could not have happened without the dedicated efforts of many people and the broad support of many more. I feel an odd combination of blessed and remorseful that I cannot possibly list everyone.

I am infinitely grateful to my Advisor, Andrew McCallum, who not only inspired this work, but also me, through his perpetual insight, enthusiasm, and encouragement. I have had the extraordinary fortune of having each and every member of my committee as a mentor and collaborator from early on in my PhD pursuit (Andrew McCallum, September, 2014; Alex Schwing, April, 2016; Andrew McGregor, July, 2016; Arya Mazumdar, April, 2017; Pat Flaherty, November, 2017) including many meetings and discussions about the trellis, algorithms for computing the partition function,

[1]which, from one perspective, is largely a study of how to count.

correlation clustering, and many other topics. Each has been generous with me beyond measure, walking me through related work, examples, and proofs, and teaching me about how to conduct informative experiments, the writing process, and research in general.

I am deeply indebted to Mark Przybocki and Chuck Romine, who enthusiastically supported my thesis efforts from concept through fruition and enabled me to pursue a PhD in machine learning.

I also feel truly thankful for: Ari Kobren and Nick Monath, who labored along side me from the very beginning, sharing with me many teas and coffees and practical insights. All of IESL, especially my direct predecessors, who were my friends, teachers, and guides: David Belanger, Arvind Neelakantan, Luke Vilnis, Emma Strubell, Pat Verga, and Ari Kobren (and a special thanks to Ben Roth, Adam Saunders, and Dan Parker). Sebastian Macaluso and Kyle Cranmer, who have been truly fantastic collaborators. All my teachers, whose encouragement helped foster in me a love of learning, especially Andrew McCallum, Mitch Marcus, Sudipto Guha, Andre Scedrov, Scott Weinstein, and Bill Wiggins. My other mentors, particularly Jack Godfrey, George Doddington, and Alvin Martin, as well as Lee Oppenheim, Darrel Whitcomb, Bob Lund, Frank Shoemaker, Phil Sternberg, Roy Holder, and John Frederick. The people who studied math and computer science with me, including Peter Fontana, Josh Magarick, Paul Cohen, Kate Silverstein, and Trapit Bansal. Eileen Hamel, Malaika Ross, and Leeanne Leclerc for keeping me from running off the rails, and for Barb Sutherland for always being happy to see me and point me in the right direction. The NIST admin staff, everyone in MIG, and the speaker and language recognition communities.

And of course I must thank my family and friends who are as family, who have supported me always. My mom, dad, sister, Aunt Chris, Uncle Wayne, Cousins Alysa, and Brian, Uncle Jim, and Aunt Lauren, for having always been a joyous part of my

life. Peter Fontana, for being a wonderful friend, a great discussion partner, and a kind soul. Andrew Oppenheim, for still being my friend after 33 years. Marc Popkin for comedy and craft services.

ABSTRACT

COMPACT REPRESENTATIONS OF UNCERTAINTY IN CLUSTERING

FEBRUARY 2021

CRAIG GREENBERG

B.M.A., VANDERBILT UNIVERSITY

B.A., UNIVERSITY OF PENNSYLVANIA

M.Sc., JOHNS HOPKINS UNIVERSITY

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew McCallum

Flat clustering and hierarchical clustering are two fundamental tasks, often used to discover meaningful structures in data, such as subtypes of cancer, phylogenetic relationships, taxonomies of concepts, and cascades of particle decays in particle physics. When multiple clusterings of the data are possible, it is useful to represent uncertainty in clustering through various probabilistic quantities, such as the distribution over partitions or tree structures, and the marginal probabilities of subpartitions or subtrees.

Many compact representations exist for structured prediction problems, enabling the efficient computation of probability distributions, e.g., a trellis structure and corresponding Forward-Backward algorithm for Markov models that model sequences. However, no such representation has been proposed for either flat or hierarchical

clustering models. In this thesis, we present our work developing data structures and algorithms for computing probability distributions over flat and hierarchical clusterings, as well as for finding maximum a posteriori (MAP) flat and hierarchical clusterings, and various marginal probabilities, as given by a wide range of energy-based clustering models.

First, we describe a trellis structure that compactly represents distributions over flat or hierarchical clusterings. We also describe related data structures that represent approximate distributions. We then present algorithms that, using these structures, allow us to compute the partition function, MAP clustering, and the marginal probabilities of a cluster (and sub-hierarchy, in the case of hierarchical clustering) exactly. We also show how these and related algorithms can be used to approximate these values, and analyze the time and space complexity of our proposed methods. We demonstrate the utility of our approaches using various synthetic data of interest as well as in two real world applications, namely particle physics at the Large Hadron Collider at CERN and in cancer genomics. We conclude with a brief discussion of future work.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

INTRODUCTION

Probabilistic models provide a rich framework for expressing and analyzing uncertain data because they provide a full joint probability distribution rather than an uncalibrated score or point estimate. There are many well-established, simple probabilistic models, for example Hidden Markov Models (HMMs) for modeling sequences. Inference in HMMs is performed using the forward-backward algorithm, which relies on an auxiliary data structure called a trellis (a graph-based dynamic programming table). This trellis structure serves as a compact representation of the distribution over state sequences. Many model structures compactly represent distributions and allow for efficient exact or approximate inference of joint and marginal distributions.

(Flat) clustering is a fundamental unsupervised learning task, used in myriad applications. Classic clustering algorithms and even modern ones, however, only provide a point estimate of the "best" partitioning according to some metric. While this is sufficient in some cases, many applications require a method to quantify the uncertainty in the provided point estimate, e.g., when selecting a clusterings from among datasets to present a human labeler, it is necessary that the clustering scores in each dataset be calibrated to enable the selection. Furthermore, normalized probabilities often prove to be beneficial over point estimates even in applications where point estimates are technically sufficient, e.g., [64]. In addition, in many applications, there are other partitions of the data that are nearly as good as the best one. Therefore representing uncertainty in clustering can allow one, for example, to chose the most interpretable clustering from among a set of nearly equivalent options.

Similarly, hierarchical clustering algorithms are used to discover meaningful structures, such as phylogenetic trees of organisms [44], taxonomies of concepts [17],

subtypes of cancer [65], and jets in particle physics [10]. 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. Additionally, hierarchical clusterings can be used to represent alternative flat partitions of a dataset.

Representing discrete distributions can be rather challenging, since the size of the support of the distribution can grow extremely rapidly. In the case of HMMs, the number of sequences that need to be represented is exponential in the sequence length. Despite this, the forward-backward algorithm (i.e., belief-propagation in a non-loopy graph) performs exact inference in time linear in the size of the sequence multiplied by the square of the size of the state space. In the case of clustering, the problem is far more difficult. The number of (flat) clusterings of N elements, known as the Nth Bell number [2], grows super-exponentially in the number of elements to be clustered. For example, there are more than a billion ways to cluster 15 elements. The case, of hierarchical clustering is worse still, with the number of hierarchies of N elements equal to $(2N-3)!!$, resulting in more than 200 trillion ways to create hierarchies of 15 elements. An exhaustive approach would require enumerating and scoring each clustering.

In this thesis, we present data structures for computing probability distributions and inference in clustering models: the cluster trellis, sparse trellis, and slab tree. We also present algorithms that use these structures to compute probability distributions over flat and hierarchical clusterings, as well as for finding maximum *a posteriori* (MAP) flat and hierarchical clusterings, and various marginal probabilities, as given by a wide range of energy-based clustering models. Using these data structures and algorithms, we are able to compute each of these various values *exactly*. Something appealing about exact solutions is that they actually solve the problem as posed. Exact solutions also have the nice property that no approximation method can ever

output a better answer, no matter how long the approximation method is allowed to run. It's also worth noting that there are applications where solving the problem exactly is of extreme benefit, for example when evaluating data models. For cases where approximations will do, it is also possible to use the presented data structures and algorithms to provide approximations of these values, which we also describe in this thesis.

The *cluster trellis*, or simply *trellis*, on a dataset, X, is a directed acyclic graph (DAG) where there is a vertex for each element of the power set of X, $\mathbb{P}(\mathrm{X})$. There is an edge between pairs of vertices, $\mathbb{V}_i$ and $\mathbb{V}_j$ if the cluster corresponding to $\mathbb{V}_i$ is a maximal strict subset of the cluster corresponding to $\mathbb{V}_j$.

The cluster trellis can be sparsified by leaving out some of the vertices and edges, which can be desirable for efficiency or to encode hard constraints. We refer to any such trellises as a *sparse trellis*. The tree structures used in hierarchical clustering are a commonly known example of a sparse trellis, which we refer to as *tree-structured sparse trellises*.

The *slab tree* is a data structure we've developed to approximate distributions over clusterings, inspired by spike and slab models. It is a tree-structured sparse trellis where each vertex is augmented with a "slab" value. These slab values are used to assign probability mass to clusters not present in the tree. We demonstrate that this substantially improves a tree-structured sparse trellis's ability to faithfully represent a distribution over clusterings. Interestingly, the slab tree is able to take an approximation of the partition function, and assign approximate probabilities to clusterings such that sum of these approximate values still sum to 1.

Our algorithms for computing the partition function, MAP clustering, and various marginals using cluster trellises and sparse trellises consist of dynamic programs that can operate in either a top-down or bottom up fashion on a cluster trellis or sparse trellis, labeling vertices with local partition functions and maximum values. In the

case of flat clusterings, it is also possible to read from these structures the likely splits and joins of clusters, as well as the marginal probability of a either a specific cluster, or of a specified set of elements being clustered together. In the case of hierarchical clusterings, the marginals that can be computed with these structures consist of the marginal probability of a given sub-hierarchy as well as of a given cluster. These algorithms work in any circumstance where the potential of a cluster can be computed. We prove that our algorithms return exact values in the case of cluster trellises. When a cluster trellis has been sparsified for efficiency reasons, our algorithms provide approximations, in a manner analogous to using beam search [60] in HMMs. When the vertices in a sparse trellis represents all clusters considered possible (i.e., any vertex missing from the sparse trellis has a corresponding cluster with zero probability, e.g., due to hard constraints or other reasons) our algorithms are again exact. We provide an analysis of their time and space complexity using the cluster trellis as well as using sparse trellises (in which case the time and space complexity are measured in the size of the sparse trellis).

Our inference algorithm using slab trees, inspired by the spike-and-slab model [51] that provides a mixture of a flat distribution with a modal distribution, works by quantizing the mass of various sub-clusterings represented by the vertices in the slab tree. Any clustering that is not present in the slab tree is given probability mass according to a slab value in the structure, determined by the correspondence between the splits represented in the tree and the splits represented in the clustering.

We also provide a few simple extensions to the presented data structures and algorithms in order to enable clustering under various natural constraints, e.g., those of the following form: two elements cannot be in the same cluster (except at the root of a hierarchical clustering). Note that it is possible to use constraints to encode existing knowledge into the clustering problem, which is useful, for example, in the physical sciences, where there is a certain amount known through existing

theory and experiments. Encoding this information can helpfully reduce the state space considered and encode distributions over clusterings that do not violate some fundamental assumptions in the domain.

We demonstrate the effectiveness of our approaches using various synthetic data of interest as well as in two real world applications, namely particle physics at the Large Hadron Collider at CERN and in cancer genomics. In the case of flat clustering, we experiment using synthetic data laid out in a grid, the UCI Zoo dataset, and data from the Cancer Genome Atlas (TCGA). For hierarchical clustering, our experiments use synthetic data generated in a manner proposed by [14] for Dasgupta cost, as well as the Prediction Analysis of Microarray 50 (pam50) gene expression dataset for heirarchical correlation clustering, and data produced by Ginkgo, a toy generative model for jets [21]. For slab trees, in addition to using UCI Zoo and TCGA data, we use a scientific author entity resolution dataset described in [80].

The compact representation of probability distributions over flat and hierarchical clusterings are fundamental problems in managing uncertainty. This thesis presents data structures and algorithms for exact and approximate inference in flat and hierarchical clustering, reducing the time complexity of these problems from super exponential in the number of data points to sub-quadratic in the size of the cluster trellis.

CHAPTER 1

BACKGROUND

1.1 Probabilistic Models and Uncertainty in Flat Clustering

Clustering[1] is the task of dividing a dataset into disjoint sets of elements. Formally,

Definition 1. (Clustering) *Given a dataset of elements, $X = \{x_i\}_{i=1}^N$, a **clustering** is a set of subsets, $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ such that $C_i \subseteq X$, $\bigcup_{i=1}^K C_i = X$, and $C_i \cap C_j = \emptyset$ for all $C_i, C_j \in \mathcal{C}$, $i \neq j$. Each element of clustering $\mathcal{C}$ is referred to as a cluster.*

Our goal is to design data structures and algorithms for efficiently computing the probability distribution over all clusterings of X. We adopt an energy-based probability model for clustering, where the probability of a clustering is proportional to the product of the potentials of the individual clusters making up the clustering. The primary assumption in energy-based clustering is that there exists a potential function mapping each clustering onto the positive reals, and that the potential function is decomposable as the product of cluster potentials. While it is intuitive that the probability of elements being clustered together would be independent of the clustering of elements disjoint from the cluster, one could conceive of distributions that violate this assumption. A decomposable potential also means that representing multiple "orthogonal" ways of clustering the data might result in relatively high probability for potentially unintuitive clusterings. For example, if an potential function represents

[1]Clustering is sometimes referred to as "flat clustering" in order to distinguish it from hierarchical clustering. We adopt this same terminology, but sometimes refer to flat clustering simply as clustering when there's no ambiguity.

clustering objects according to color and shape, clusterings that cluster some elements according to color and others according to shape will necessarily have will have relatively high probability mass, perhaps greater than what might be expected if a human were to assign a relative probability to these clusterings. An additional assumption is that exponentiating pairwise scores preserves item similarity. This is the Gibbs distribution, which has been found useful in practice [29].

Definition 2. (Energy Based Clustering) *Let X be a dataset, $\mathcal{C}$ be a clustering of X, and $\mathcal{E}_X(\mathcal{C})$ be the potential function describing the compatibility of a clustering $\mathcal{C}$, and $\mathcal{E}_X(C)$ be the potential for cluster C. Then, the probability of $\mathcal{C}$ with respect to X, $P_X(\mathcal{C})$, is equal to the potential of $\mathcal{C}$ normalized by the partition function, $Z(X)$. This gives us $P_X(\mathcal{C}) = \frac{\mathcal{E}_X(\mathcal{C})}{Z_X}$ and $Z(X) = \sum_{\mathcal{C}\in\mathbb{C}_X} \mathcal{E}_X(\mathcal{C})$. The potential of clustering $\mathcal{C}$,is defined as the product of the potentials of its clusters: $\mathcal{E}_X(\mathcal{C}) = \prod_{C\in\mathcal{C}} \mathcal{E}_X(C)$.*

We refer to this as an energy-based model since often it is the case that $\mathcal{E}_X(\cdot)$ is defined by the unnormalized Gibbs distribution as $\mathcal{E}_X(\mathcal{C}) = \exp(-\beta E_X(\mathcal{C}))$, where β is the inverse temperature and $E_X(\mathcal{C})$ is the energy.

We use $\mathbb{C}_X$ to refer to all clusterings of X. In general, we assume that X is fixed and so we omit subscripts to simplify notation. Note that computing the membership probability of any element x_i in any cluster C_j, as is done in mixture models, is ill-suited for our goal. In particular, this computation assumes a fixed clustering whereas our work focuses on computations performed with respect to the distribution over all possible clusterings.

1.2 Probabilistic Models and Uncertainty in Hierarchical Clustering

A hierarchical clustering is a recursive splitting of a dataset 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). This is in contrast to flat clustering, where the task is to partition the dataset into disjoint subsets. Formally,

Definition 3. (Hierarchical Clustering[2]) *Given a dataset of elements,* $X = \{x_i\}_{i=1}^N$*, a* ***hierarchical clustering****,* $\mathtt{H}$*, 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.* $X_j \bigcup X_k = X_i$*.*

When a subset $X_j \in \mathtt{H}$, X_j is referred to as a cluster in $\mathtt{H}$. When $X_i, X_j, X_k \in \mathtt{H}$ and $X_j \bigcup X_k = X_i$, we refer to X_j and X_k as children of X_i, and X_i the parent of X_j and X_k; if $X_j \subset X_i$ we refer to X_i as an ancestor of X_j and X_j a descendent of X_i. Leaves of the tree refer to individual elements / singleton clusters.

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. Our model is based on measuring the compatibility of all pairs of sibling nodes in a binary tree structure. Formally,

Definition 4. (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* $\phi(\mathtt{H})$ *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,* $Z(X)$*:*

$$P(\mathtt{H}|X) = \frac{\phi(\mathtt{H})}{Z(X)} \quad \textit{where} \quad \phi(\mathtt{H}) = \prod_{X_L, X_R \in \textit{sibs}(\mathtt{H})} \psi(X_L, X_R) \tag{1.1}$$

[2]We limit our exposition to binary hierarchical clustering. Binary structures encode more tree-consistent clusterings than k-ary [8]. Natural extensions may exist for k-ary clustering, which are left for future work.

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

$$Z(X) = \sum_{\mathtt{H} \in \mathcal{H}(X)} \phi(\mathtt{H}). \tag{1.2}$$

where $\mathcal{H}(X)$ *gives all binary hierarchical clusterings of the elements* X*. We refer to this as an energy-based model since often it is the case that* $\psi(\cdot, \cdot)$ *is defined by the unnormalized Gibbs distribution, as* $\psi(X_L, X_R) = \exp(-\beta E(X_L, X_R))$*, 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. We now give three example instances of the aforementioned probabilistic model, each corresponding to a well-known or otherwise important use cases:

Use Case 1. ***(Jet Physics)*** 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, where for each pair of parent and left (right) child clusters, with masses $\sqrt{t_P}$ and $\sqrt{t_L}$ ($\sqrt{t_R}$) respectively, the likelihood function is,

$$\psi(X_L, X_R) = f(t(x_L)|t_P, \lambda) \cdot f(t(x_R)|t_P, \lambda) \tag{1.3}$$

where

$$f(t|t_P, \lambda) = \frac{1}{1 - e^{-\lambda}} \frac{\lambda}{t_P} e^{-\lambda \frac{t}{t_P}} \tag{1.4}$$

The first term is a normalization factor associated to the constraint that $t < t_P$.

Use Case 2. ***(Hierarchical Correlation Clustering)*** In the cancer genomics use case (§5.2.2), we are given a dataset of vectors indicating level of gene expressions which are endowed with pairwise affinities that are both positive and negative. In this case, we define the potential of a pair of sibling nodes in the tree to be the sum of the positive edges not crossing the cut, minus the sum of the negative edges crossing the cut:

$$\psi(X_i, X_j) = \exp(-\beta E(X_i, X_j)) \tag{1.5}$$

$$E(X_i, X_j) = \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \mathbb{I}[w_{ij} > 0] - \sum_{\substack{x_i, x_j \in X_i \times X_i, \\ x_i \neq x_j}} w_{ij} \mathbb{I}[w_{ij} < 0] - \sum_{\substack{x_i, x_j \in X_j \times X_j, \\ x_i \neq x_j}} w_{ij} \mathbb{I}[w_{ij} < 0] \tag{1.6}$$

where w_{ij} is the affinity between x_i and x_j. This potential is the correlation clustering objective [5].

Use Case 3. ***(Dasgupta's Cost)*** Dasgupta [24] defines a cost function for hierarchical clustering that has been the subject of much theoretical interest (primarily on approximation algorithms for the cost) [18, 19, 12, 14, 53, 62]. 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:

$$E(X_i, X_j) = (|X_i| + |X_j|) \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \tag{1.7}$$

In [24], Dasgupta gives two equivalent formulations of the cost function, and Equation 1.7 corresponds to the splitting cost in the cut-cost formulation, where the weight of the cut is given by $\sum_{x_i, x_j \in X_i \times X_j} w_{ij}$.

1.3 Declaration of Collaborations and Previous Work

The work in this thesis is either published, pending submission, in submission, or, in the case of future work, is active research, and is a joint effort with the named researchers. Some of the text in this thesis is directly from the following:

- Greenberg, C., Monath, N., Kobren, A., Flaherty, P., McGregor, A., & McCallum, A. (2018). "Compact Representation of Uncertainty in Clustering". In *Advances in Neural Information Processing Systems* (pp. 8630-8640). [31]

- Greenberg, C., Monath, N., Kobren, A., Flaherty, P., McGregor, A., & McCallum, A. "Distributions over Clusterings using Slab Trees". [Submission Pending]

- *Greenberg, C., *Macaluso, S., Monath, N., Lee, J., Flaherty, P., Cranmer, K., McGregor, A., & McCallum, A. "Compact Representation of Uncertainty in Hierarchical Clustering". [Submitted]

- Macaluso, S., Monath, N., Greenberg, C., & Cranmer, K. "A Sparse Trellis for Approximate Inference on Hierarchical Clusterings". [Submission Pending]

- *Greenberg, C., *Monath, N., Macaluso, S., Dubey, A., Zaheer, M., Ahmed, A., Flaherty, P., Cranmer, K., & McCallum, A. "Exact and Approximate Inference in Hierarchical Clustering Using A*". [Work in Progress]

CHAPTER 2

DATA STRUCTURES FOR COMPUTING PROBABILITY DISTRIBUTIONS AND MAP ESTIMATES OF CLUSTERING MODELS

2.1 The Cluster Trellis

To support the computation of probabilistic quantities in clustering models, we introduce an auxiliary data structure we call a *cluster trellis*.

Definition 5. (Cluster Trellis) *A cluster trellis, $\mathbb{T}$, over a dataset of elements, $X = \{x_i\}_{i=1}^{N}$, is a graph, $\mathbb{T} = (\mathbb{V}(\mathbb{T}), E(\mathbb{T}))$, whose vertices represent all valid clusters of elements of X. The edges of the graph connect a pair vertices if one (the "child" node) is a maximal strict subset of the other (the "parent" node).*

More formally, the cluster trellis data structure is a directed acyclic graph (DAG), where there is a bijection between the vertices of the graph and the power set of dataset X sans the empty set, i.e., $\mathbb{P}(X) \setminus \emptyset$. This bijection associates with each trellis vertex $\mathbb{V}$ a dataset, denoted $X(\mathbb{V})$, while the trellis vertex associated with a dataset X is denoted $\mathbb{V}(X)$. There exists an edge from vertex $\mathbb{V}_i$ to $\mathbb{V}_j$, if $\exists \mathbb{V}_k \in \mathbb{V}(\mathbb{T})$ s.t. the set of elements associated with $\mathbb{V}_j$ and the set of elements associated with $\mathbb{V}_i$ have an empty intersection and a union equal to the set of elements associated with $\mathbb{V}_k$ (i.e., $X(\mathbb{V}_i) \bigcap X(\mathbb{V}_j) = \emptyset \wedge X(\mathbb{V}_i) \bigcup X(\mathbb{V}_j) = X(\mathbb{V}_k)$).

We note that the edges of the structure are not crucial for the presentation given in this thesis of our algorithms, with the exception of algorithms for approximating distributions over hierarchical clusterings. Nevertheless they are practically useful in the implementation of the algorithms.

In this thesis, we refer to a cluster trellis (see Definition 5) simply as a *trellis*[1]. Each vertex in the trellis, $\mathbb{V} \in \mathbb{V}(\mathbb{T})$, stores various memoized values, including a partition function sub-value $Z(X(\mathbb{V}))$, used for computing the partition function. For flat clustering, $\mathbb{V}$ also stores the potential of the associated data set, $\mathcal{E}(X(\mathbb{V}))$, as well as the MAP clustering value and MAP clustering of $X(\mathbb{V})$. For hierarchical clustering, the MAP tree value, $\phi(\mathtt{H}^*[\mathbb{V}])$ and the backpointer $\Xi(\mathtt{H}^*[\mathbb{V}])$ for the MAP tree.

See Figure 2.1 for a visual representation of a clustering trellis over 4 elements. Each vertex memoizes a potential of its associated cluster, $\mathcal{E}(X(\mathbb{V}))$, each of which can be computed as a function over a constant number outbound edges. We borrow terminology from DAGs and say vertex $\mathbb{V}'$ is a *parent* of vertex $\mathbb{V}$, if there is an edge from $\mathbb{V}'$ to $\mathbb{V}$, and that vertex $\mathbb{V}''$ is an *ancestor* of $\mathbb{V}$ if there is a directed path from $\mathbb{V}''$ to $\mathbb{V}$.

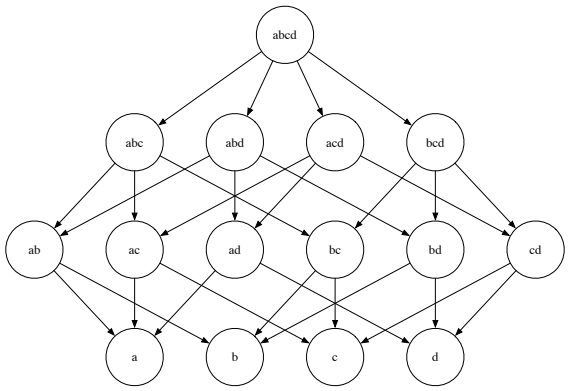


Figure 2.1: A cluster trellis, $\mathbb{T}$, over a dataset $X = \{a, b, c, d\}$. Each node in the trellis represents a specific cluster, i.e., subset, of X corresponding to its label. Solid lines indicate parent-child relationships. Note that a parent may have multiple children and a child may have multiple parents.

[1]Note the cluster trellis structure as described in Definition 5 is closely related to the Hasse Diagram [73], though has several important distinctions: (1) the trellis is a data structure that allows memoization and other programming techniques, (2) trellises can be sparsified, removing some of the vertices and edges (see Section 2.2), and (3) trellises permit other edge arrangements (see, for example, the hierarchical clustering sparse trellis edge arrangements in Definition 6).

2.2 The Sparse Trellis

Unfortunately, the size of the trellis scales exponentially with the size of the dataset, which limits the use of the trellis in practice. In this section, we introduce the *sparse trellis*, which is a trellis with some nodes and edges omitted. Increasing the sparsity of a trellis enables the computation of clustering distributions for larger datasets.

Definition 6. (Sparse Trellis) *Given a trellis* $\mathbb{T} = (\mathbb{V}(\mathbb{T}), E(\mathbb{T}))$*, we define a sparse trellis with respect to* $\mathbb{T}$ *to be any* $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}), E(\widehat{\mathbb{T}}))$ *where* $\mathbb{V}(\widehat{\mathbb{T}}) \neq \emptyset$*,* $\mathbb{V}(\widehat{\mathbb{T}}) \subset \mathbb{V}(\mathbb{T})$*, and* $E(\widehat{\mathbb{T}}) \subset E(\mathbb{T})$*.*

Definition 7. (Sparse Trellis for Flat Clustering) *A sparse trellis for flat clustering is a sparse trellis* $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}), E(\widehat{\mathbb{T}}))$*, where the edges satisfy the following additional property:* $E(\widehat{\mathbb{T}}) = \{(\mathbb{V}, \mathbb{V}') |\ X(\mathbb{V}') \subset X(\mathbb{V}) \wedge \nexists \mathbb{V}'' \in \mathbb{V}(\widehat{\mathbb{T}}) : X(\mathbb{V}') \subset X(\mathbb{V}'') \subset X(\mathbb{V})\}$*;*

Definition 8. (Sparse Trellis for Hierarchical Clustering) *A sparse trellis for hierarchical clustering is a sparse trellis* $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}), E(\widehat{\mathbb{T}}))$*, where the edges satisfy the following additional property:* $E(\widehat{\mathbb{T}}) \subseteq \{(\mathbb{V}, \mathbb{V}') |\ X(\mathbb{V}') \subset X(\mathbb{V}) \wedge \exists \mathbb{V}'' \in \widehat{\mathbb{T}} : X(\mathbb{V}') \bigcap X(\mathbb{V}'') = \emptyset \wedge X(\mathbb{V}') \bigcup X(\mathbb{V}'') = X(\mathbb{V})\}$*.*

We sometimes refer to a trellis as a "full" trellis, in contrast to a sparse trellis, if it contains all valid vertices and edges over a dataset.

The key challenge of analyzing a sparse trellis, $\widehat{\mathbb{T}}$, is how to treat any cluster $C \in \mathbb{P}(X) \setminus \emptyset$ that is not represented by a vertex in the sparse trellis, i.e., $C \notin \{X(\mathbb{V}) : \mathbb{V} \in \mathbb{V}(\widehat{\mathbb{T}})\}$. Although there are several feasible approaches to reasoning about such clusters, in this thesis we assume that any cluster that is not represented by a vertex in a sparse trellis has zero potential, consistent with how missing clusters are treated when using trees to represent sets of potential flat clusterings. We address this differently when defining slab tree structures (see Section 2.3).

Since the potential of a clustering, $\mathcal{C}$, is the product of its clusters' potentials (Definition 2), $\mathcal{E}(\mathcal{C}) = 0$ if it contains one or more clusters with zero potential.

2.2.1 Sparse Trellises Containing Valid Clusterings

Note that there exist a number of sparse trellises that contain no valid clusterings. As an example, consider $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}) = \{\mathbb{V}_1, \mathbb{V}_2, \mathbb{V}_3\}, E(\widehat{\mathbb{T}}) = \emptyset)$, where $X(\mathbb{V}_1) = \{x_1, x_2\}$, $X(\mathbb{V}_2) = \{x_2, x_3\}$, and $X(\mathbb{V}_3) = \{x_1, x_3\}$.

For ease of analysis, with the exception of tree-structured sparse trellises, in flat clustering we focus on a specific family of sparse trellises which are closed under recursive complement.

Definition 9. (Closed Under Recursive Complement) *A sparse trellis* $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}), E(\widehat{\mathbb{T}}))$*, is closed under recursive complement iff:* $\forall \mathbb{V}, \mathbb{V}' \in \mathbb{V}(\widehat{\mathbb{T}}), X(\mathbb{V}) \subset X(\mathbb{V}') \implies \exists \mathbb{V}'' \in \mathbb{V}(\widehat{\mathbb{T}}) : X(\mathbb{V}) \bigcup X(\mathbb{V}'') = X(\mathbb{V}') \wedge X(\mathbb{V}) \cap X(\mathbb{V}'') = \emptyset$.

This property ensures that the trellises contain only valid partitions.

Similarly, in hierarchical clustering, we restrict our consideration to trellises that only contain valid hierarchies, namely $\forall \mathbb{V}, \mathbb{V}' \in \mathbb{V}(\widehat{\mathbb{T}})$, if there exists an edge from $\mathbb{V}$ to $\mathbb{V}'$, there must exist a $\mathbb{V}'' \in \mathbb{V}(\widehat{\mathbb{T}})$ s.t. $X(\mathbb{V}') \bigcap X(\mathbb{V}'') = \emptyset$, $X(\mathbb{V}') \bigcup X(\mathbb{V}'') = X(\mathbb{V})$, and there exists and edge from $\mathbb{V}$ to $\mathbb{V}''$. Note that this is a less restrictive condition than being closed under recursive complement.

2.2.2 Tree-Structured Sparse Trellis

The often-used hierarchical (tree-structured) clustering encompasses another family of sparse trellises. Each vertex in a tree-structured trellis has at most one parent. A tree-structured trellis meets the definition of a sparse trellis since, for any two vertices, $\mathbb{V}_1, \mathbb{V}_2$, in such a trellis, exactly one of the following must hold: $X(\mathbb{V}_1) \subset X(\mathbb{V}_2), X(\mathbb{V}_2) \subset X(\mathbb{V}_1)$, or $X(\mathbb{V}_1) \cap X(\mathbb{V}_2) = \emptyset$.

This family of sparse trellises has the advantage that many practical algorithms can be used for trellis construction, such as hierarchical agglomerative clustering. Note that (with the exception of when $|\mathbb{V}(\widehat{\mathbb{T}})| = 1$) a tree-structured sparse trellis represents multiple flat clusterings and (always) a single hierarchical clustering.

2.3 The Slab Tree for Flat Clustering

Using tree data structures for approximating distributions over flat clusterings is convenient, since algorithms for efficient computation using trees are well-known. However, when zero probability mass is given to partitions that are not tree consistent, as is typically done when using trees to encode distributions over clusterings, the resultant distribution can be a poor approximation to the true distribution over clusterings.

In particular, the overwhelmingly vast majority of the possible clusterings are not tree consistent, and thus will be given no mass, even those containing high potential. This deficiency of using trees to encode distributions over clusterings is the fundamental motivation for our development of slab trees, which do not suffer the same loss of probability mass for non-tree consistent partitions.

Definition 10. (Slab Tree) *A slab tree, $\widehat{\mathbb{T}}$, is a tree-structured sparse trellis, where each vertex in the slab tree, $\mathbb{V}$, has an associated slab potential, $\mathcal{S}(\mathbb{V})$, representing the potential of clusterings that are not consistent with the tree structure of $\widehat{\mathbb{T}}$.*

The slab values act as cluster-based smoothing that distributes probability mass to clusterings not in the tree structure. Slab trees can have arbitrary branching factor, however we limit our focus to binary trees since they simultaneously encode the largest number of tree-consistent partitions, and are also common in practice [33, 8, 41].

CHAPTER 3

EXACT DISTRIBUTIONS OVER FLAT CLUSTERINGS

3.1 Algorithms and Analysis

Recall that our goal is to efficiently compute a distribution over the valid clusterings of an instance of energy-based clustering. Given a dataset X, a naïve first step in computing such a distribution is to iterate through its unique clusterings and, for each, compute its potential and add it to a running sum. If the number of elements is $|X| = N$, the number of unique clusterings is the N^{th} Bell Number, B_N, expressible as the recursive formula $B_N = \sum_{k=0}^{N-1} \binom{N-1}{k} B_k$, which is super-exponential in N [47].

Note that a cluster, C, may appear in many clusterings of X. For example, consider the dataset $X' = \{a, b, c, d\}$. The cluster $\{a, b\}$ appears in 2 of the clusterings of X'. More precisely, in a dataset comprised of N elements, a cluster of M elements appears in the $(N - M)^{th}$ Bell Number of its clusterings. This allows us to make use of memoization to compute the distribution over clusterings more efficiently, in a procedure akin to variable elimination in graphical models [27, 79]. Unlike variable elimination, our procedure is agnostic to the ordering of the elimination.

3.1.1 Computing the Partition Function

Computing a distribution over an event space requires computing a partition function, or normalizing constant. We present an algorithm for computing the partition function, $Z(X)$, with respect to all possible clusterings of the elements of X, i.e., $\mathbb{C}$. Our algorithm uses the trellis and a particular memoization scheme to significantly reduce the computation required: from super-exponential to exponential.

The full partition function, $Z(X)$, can be expressed in terms of cluster potentials and the partition functions of a specific set of *subtrellises.* A subtrellis rooted at $\mathbb{V}$, denoted $\mathbb{T}[\mathbb{V}]$ contains all nodes in $\mathbb{T}$ that are descendants of $\mathbb{V}$.

Formally, a *subtrellis*, $\mathbb{T}[\mathbb{V}] = (\mathbb{V}(\mathbb{T}[\mathbb{V}]), E(\mathbb{T}[\mathbb{V}]))$, has vertices and edges satisfying the following properties: (1) $\mathbb{V}(\mathbb{T}[\mathbb{V}]) = \{\mathbb{V}' \in \mathbb{V}(\mathbb{T}) | X(\mathbb{V}') \subseteq X(\mathbb{V})\}$, and (2) $E(\mathbb{T}[\mathbb{V}]) = \{(\mathbb{V}', \mathbb{V}'') | (\mathbb{V}', \mathbb{V}'') \in E(\mathbb{T}) \wedge \mathbb{V}', \mathbb{V}'' \in \mathbb{V}(\mathbb{T}[\mathbb{V}])\}$. Note that $\mathbb{T}[\mathbb{V}]$ always forms a valid trellis.

The following procedure not only computes $Z(X)$, but also generalizes in a way that the partition function with respect to clusterings for any subset $X' \subset X$, i.e., $Z(X')$, can also be computed. We refer to the partition function for a dataset $X(\mathbb{V})$ memoized at the trellis/subtrellis $\mathbb{T}[X(\mathbb{V})]$ as the partition function for the trellis/subtrellis, $Z(\mathbb{T}[X(\mathbb{V})])$.

Algorithm 1 `PartitionFunction`$(\mathbb{T}, X)$

```
Pick x_i ∈ X
Z(X) = 0
for 𝕍 in 𝕍(𝕋)^(i) do
    Let 𝕍' be such that X(𝕍') = X \ X(𝕍)
    if Z(X(𝕍')) has not been assigned then
        Z(X(𝕍')) = PartitionFunction(𝕋[𝕍'], X(𝕍'))
    Z(X) ← Z(X) + ℰ(X(𝕍)) * Z(X(𝕍'))
return Z(X)
```

Define $\mathbb{V}(\mathbb{T})^{(i)} = \{\mathbb{V} \in \mathbb{V}(\mathbb{T}) | x_i \in X(\mathbb{V})\}$ and $\overline{\mathbb{V}(\mathbb{T})^{(i)}} = \mathbb{V}(\mathbb{T}) \backslash \mathbb{V}(\mathbb{T})^{(i)}$. In other words, $\mathbb{V}(\mathbb{T})^{(i)}$ is the set of all vertices in the trellis containing the element x_i and $\overline{\mathbb{V}(\mathbb{T})^{(i)}}$ is the set of all vertices that do not contain x_i.

Proposition 1. *Let $\mathbb{V} \in \mathbb{V}(\mathbb{T})$ and $x_i \in X(\mathbb{V})$. The partition function with respect to $X(\mathbb{V})$ can be written recursively, with $Z(X(\mathbb{V})) = \sum_{\mathbb{V}_i \in \mathbb{V}(\mathbb{T}[\mathbb{V}])^{(i)}} \mathcal{E}(\mathbb{V}_i) \cdot Z(X(\mathbb{V}) \backslash X(\mathbb{V}_i))$ and $Z(\emptyset) = 1$.*

Proof. The partition function $Z(X(\mathbb{V}))$ is defined as:

$$Z(X(\mathbb{V})) = \sum_{\mathcal{C} \in \mathbb{C}_{X(\mathbb{V})}} \prod_{C \in \mathcal{C}} \mathcal{E}(C)$$

For a given element x_i in $X(\mathbb{V})$, the set of all clusterings of $X(\mathbb{V})$ can be re-written to factor out the cluster containing x_i in each clustering,

$$\mathbb{C}_{X(\mathbb{V})} = \{\{\mathbb{V}_i\} \cup \mathcal{C} | v_i \in V^{(i)}, \mathcal{C} \in \mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}_i)}\}.$$

Note that $\mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}_i)}$ refers to all clusterings on the elements $X(\mathbb{V}) \setminus X(\mathbb{V}_i)$. Using this expansion and since $\mathcal{E}(\{\mathbb{V}_i\} \cup \mathcal{C}_i) = \mathcal{E}(\{\mathbb{V}_i\})\mathcal{E}(\mathcal{C}_i)$, we can rewrite the partition function as below. By performing algebraic re-arrangements and applying our definitions:

$$\begin{aligned}
Z(X(\mathbb{V})) &= \sum_{\mathbb{V}_i \in V^{(i)}} \sum_{\mathcal{C} \in \mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}_i)}} \mathcal{E}(\mathbb{V}_i)\mathcal{E}(\mathcal{C}) \\
&= \sum_{\mathbb{V}_i \in V^{(i)}} \sum_{\mathcal{C} \in \mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}_i)}} \mathcal{E}(\mathbb{V}_i) \prod_{C \in \mathcal{C}} \mathcal{E}(C) \\
&= \sum_{\mathbb{V}_i \in V^{(i)}} \mathcal{E}(\mathbb{V}_i) \sum_{\mathcal{C} \in \mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}_i)}} \prod_{C \in \mathcal{C}} \mathcal{E}(C) \\
&= \sum_{\mathbb{V}_i \in V^{(i)}} \mathcal{E}(\mathbb{V}_i) Z(X(\mathbb{V}) \setminus X(\mathbb{V}_i))
\end{aligned}$$

□

As a result of Proposition 1, we are able to construct a dynamic program for computing the partition function of a trellis that computes the exact partition function value without repeating any computations. It works as follows: (1) select an arbitrary element x_i from the dataset;

(2) construct $\mathbb{V}(\mathbb{T})^{(i)}$ as defined above;

(3) for each vertex $\mathbb{V}_i \in \mathbb{V}(\mathbb{T})^{(i)}$, compute and memoize the partition function of $X(\mathbb{V}) \setminus X(\mathbb{V}_i)$ (if it is not already memoized);

(4) sum the partition function values obtained in step (3). The pseudocode for this dynamic program appears in Algorithm 1.

We use Algorithm 1 and Proposition 1 to analyze the time and space complexity of computing the partition function. Consider a trellis $\mathbb{T}$ over a dataset $X = \{x_i\}_{i=1}^N$. Our goal is to compute the partition function, $Z(\mathbb{T})$. When the partition function of all subtrellises of $\mathbb{T}$ have already been computed, Algorithm 1 is able to run without recursion.

Proposition 2. *Let $\mathbb{T}$ be a trellis such that the partition function corresponding to each of its subtrellises, $\mathbb{T}[\mathbb{V}_i]$, is memoized and accessible in constant time. Then, $Z(\mathbb{T})$ can be computed by summing exactly 2^{N-1} terms. Given that the partition function of every strict sub-trellis of $\mathbb{T}$ (i.e., any sub-trellis of $\mathbb{T}$ that is not equivalent to $\mathbb{T}$) has been memoized and is accessible in constant time, then $Z(\mathbb{T})$ is computed by taking the sum of exactly 2^{N-1} terms.*

Proof. According to Proposition 1, $Z(\mathbb{T})$ can be written as a sum of products of cluster potentials and partition functions. Note that $\mathbb{V}(\mathbb{T})^{(i)}$ and $\overline{\mathbb{V}(\mathbb{T})^{(i)}}$ are disjoint, $\mathbb{V}(\mathbb{T})^{(i)} \cup \overline{\mathbb{V}(\mathbb{T})^{(i)}} = \mathbb{V}(\mathbb{T})$, and $\overline{\mathbb{V}(\mathbb{T})^{(i)}}$ represents the nonempty sets in the powerset of $N-1$ elements. This implies the size of $\mathbb{V}(\mathbb{T})^{(i)}$ is 2^{N-1}. Therefore, in the special case described in Proposition 2, the trellis can be used to compute the partition function in time 2^{N-1}. □

We now consider the more general case, where the partition function of all subtrellises of $\mathbb{T}$ have not yet been computed:

Theorem 1. *Let $\mathbb{T}$ be a trellis over $X = \{x_i\}_{i=1}^N$. Then, $Z(\mathbb{T})$ can be computed in $\mathcal{O}(3^{N-1}) = \mathcal{O}(|\mathbb{V}(\mathbb{T})|^{\log(3)})$ time.*

Proof. We compute $Z(\mathbb{T})$ using the equation defined in Proposition 1. To begin, an element $x_i \in X$ is chosen and $\mathbb{V}(\mathbb{T})^{(i)}$ is constructed. Computing $Z(\mathbb{T})$ requires the cluster potential of each $\mathbb{V} \in \mathbb{V}(\mathbb{T})^{(i)}$ (recall that there are 2^{N-1} such vertices) and the corresponding partition functions. These partition functions are computed for a sub-trellis $\mathbb{T}[\mathbb{V}]$ such that $x_i \notin X(\mathbb{V})$. Let $\mathbb{T}^k$ be the set of sub-trellises of $\mathbb{T}$ over k

elements, none of which are x_i. If the partition function of every sub-trellis in $\mathbb{T}^{k-1}$ is computed and memoized before the partition function of any sub-trellis in $\mathbb{T}^k$, then, for any sub-trellis $\mathbb{T}[\mathbb{V}] \in \mathbb{T}^k$, all relevant partition functions will have been memoized. By Proposition 2, computing $Z(\mathbb{T}[\mathbb{V}])$ includes exactly 2^{k-1} terms.

What remains to be analyzed is the number of sub-trellises in each set $\mathbb{T}^k$. Recall that any sub-trellis in $\mathbb{T}^k$ must not contain the element x_i. Then, the number of sub-trellises in $\mathbb{T}^k$ is $\binom{N-1}{k}$. Summing over all subtrellises, we must compute:

$$\sum_{k=1}^{N-1} \binom{N-1}{k} 2^{k-1} = \frac{1}{6}(3^N - 3)$$

terms. In total, computing the cost of the summing over all the subtrellis and the cost of computing $Z(\mathbb{T})$ as given by Proposition 2 is $\frac{1}{6}(3^N - 3) + 2^{N-1} = \mathcal{O}(3^{N-1})$. Since $|\mathbb{V}(\mathbb{T})| = 2^N$, then $\mathcal{O}(3^{N-1}) = \mathcal{O}(|\mathbb{V}(\mathbb{T})|^{\log(3)})$. □

3.1.2 Computing the MAP Clustering

By making a minor alteration to Algorithm 1, we are also able to compute the value of and find the clustering with maximal potential. Specifically, at each vertex in the trellis, $\mathbb{V}$, store the clustering of $X(\mathbb{V})$ with maximal potential (and its associated potential). We begin by showing that there exists a recursive form of the max-partition calculation analogous to the computation of the partition function in Proposition 1.

Definition 11. (MAP Clustering) *Let $\mathbb{V} \in \mathbb{V}(\mathbb{T})$ and $x_i \in X(\mathbb{V})$. The maximal clustering over the elements of $X(\mathbb{V})$, denoted $\mathcal{C}^\star(X(\mathbb{V}))$, is defined as: $\mathcal{C}^\star(X(\mathbb{V})) = \operatorname{argmax}_{\mathcal{C} \in \mathbb{C}_{X(\mathbb{V})}} \mathcal{E}(\mathcal{C})$.*

Proposition 3. *$\mathcal{C}^\star(X(\mathbb{V}))$ can be written recursively as $\mathcal{C}^\star(X(\mathbb{V})) = \operatorname{argmax}_{\mathbb{V}' \in \mathbb{V}(\mathbb{T}[\mathbb{V}])^{(i)}} \mathcal{E}(\mathbb{V}') \cdot \mathcal{E}(\mathcal{C}^\star(X(\mathbb{V}) \setminus X(\mathbb{V}')))$.*

Proof. Consider $\mathcal{C}^\star(X(\mathbb{V}))$, the clustering with the maximal potential over $X(\mathbb{V})$. Select an arbitrary element $x_i \in X(\mathbb{V})$. Since $\mathcal{C}^\star(X(\mathbb{V}))$ is a valid clustering, it must

contain only one cluster that contains x_i; call that cluster $C_i^\star$. Let cluster $C_i^\star$ be represented by a node $\mathbb{V}' \in \mathbb{V}(\mathbb{T}[\mathbb{V}])$. Given $C_i^\star$, we can construct $\mathcal{C}^\star(X(\mathbb{V}))$ by finding the maximal clustering of the remaining elements, i.e.,

$$\operatorname*{argmax}_{\mathbb{V}'' \in \mathbb{C}_{X(\mathbb{V}) \setminus X(\mathbb{V}')}} \mathcal{E}(X(\mathbb{V}'')) = \mathcal{E}(\mathcal{C}^\star(X(\mathbb{V}) \setminus X(\mathbb{V}')))$$

Finally, $C_i^\star \in \mathcal{C}^\star(X(\mathbb{V}))$ since we take the argmax with the respect to $\mathbb{V}(\mathbb{T}[\mathbb{V}])^{(i)}$ and $C_i^\star \in \mathbb{V}(\mathbb{T}[\mathbb{V}])^{(i)}$. □

In other words, the clustering with maximal potential over the set of elements, $X(\mathbb{V})$ can be written as the potential of any cluster, C, in that clustering multiplied by a clustering with maximal potential over the elements $X(\mathbb{V}) \setminus C$.

Using this recursive definition, we modify Algorithm 1 to compute the MAP clustering instead of the partition function, resulting in Algorithm 2.

Algorithm 2 `MAPClustering`$(\mathbb{T}, X)$

```
Pick x_i ∈ X
MaxScore(X) ← 0
MaxPart(X) ← ∅
for 𝕍 in 𝕍(𝕋)^(i) do
  Let 𝕍′ be such that X(𝕍′) = X \ X(𝕍)
  if MaxScore(X(𝕍′)) has not been assigned then
    MaxCluster(T[𝕍′], X(𝕍′))
    if MaxScore(X) < ℰ(X(𝕍)) · MaxScore(X(𝕍′)) then
      MaxScore(X) = ℰ(X(𝕍)) · MaxScore(X(𝕍′))
      MaxPart(X) = X(𝕍) ∪ MaxPart(X(𝕍′))
return MaxPart(X),MaxScore(X)
```

The correctness of this algorithm is demonstrated by Proposition 3. We can now analyze the time complexity of the algorithm. We use similar memoized notation for the algorithm where $\mathcal{C}^\star(\mathbb{T}[X(\mathbb{V})])$ is the memoized value for $\mathcal{C}^\star(X(\mathbb{V}))$ stored at $\mathbb{V}$.

Proposition 4. *Let $\mathbb{T}$ be a trellis over $X = \{x_i\}_{i=1}^N$. Then, $\mathcal{C}^\star(\mathbb{T}_X)$ can be computed in $\mathcal{O}(3^{N-1})$ time.*

Proof. Observe that the number of recursive calls to Algorithm 2 is the same as the number of recursive calls to Algorithm 1. Next, observe that the amount of computation required at each recursive call follows the same argument as in the proof of Theorem 1. □

3.1.3 Computing Marginals

The trellis facilitates the computation of two types of cluster marginals. First, the trellis can be used to compute the probability of a specific cluster, $X(\mathbb{V})$, with respect to the distribution over all possible clusterings; and second, it can be used to compute the probability that any group of elements, X, are clustered together. We begin by analyzing the first style of marginal computation as it is used in computing the second.

Let $\mathbb{C}^{(\mathbb{V})} \in \mathbb{C}$ be the set of clusterings that contain the cluster $X(\mathbb{V})$. Then the marginal probability of $X(\mathbb{V})$ is given by $P(X(\mathbb{V})) = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\mathbb{V})}} \mathcal{E}(\mathcal{C})}{Z(X)}$, where $Z(X)$ is the partition function with respect to the full trellis described in section 1.1. This probability can be re-written in terms of the complement of $X(\mathbb{V})$, i.e., $P(X(\mathbb{V})) = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\mathbb{V})}} \mathcal{E}(\mathcal{C})}{Z(X)} = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\mathbb{V})}} \mathcal{E}(X(\mathbb{V}))\mathcal{E}(\mathcal{C} \setminus X(\mathbb{V}))}{Z(X)} = \frac{\mathcal{E}(X(\mathbb{V})) \sum_{\mathcal{C}' \in \mathbb{C}_{X \setminus X(\mathbb{V})}} \mathcal{E}(\mathcal{C}')}{Z(X)} = \frac{\mathcal{E}(X(\mathbb{V}))Z(X \setminus X(\mathbb{V}))}{Z(X)}$.

Note that if $Z(X \setminus X(\mathbb{V}))$ were memoized during Algorithm 1, then computing the marginal probability requires constant time and space equal to the size of the trellis. This is only true for clusters whose complements do not contain element x_i (selected to compute $Z(X)$ in Algorithm 1), which is true for $|V(\mathbb{T})|/(2|V(\mathbb{T})| - 1)$ of the vertices in the trellis. Otherwise, we may need to repeat the calculation from Algorithm 1 to compute $Z(X \setminus X(\mathbb{V}))$. We note that due to memoization, the complexity of computing the partition function of the remaining verticies is no greater than the complexity of Algorithm 1.

This machinery makes it possible to compute the second style of marginal. Given a set of elements, X, the marginal probability of the elements of X being clustered

together is: $P(X) = \sum_{X(\mathbb{V}) \in \mathbb{T}: X \subseteq X(\mathbb{V})} P(X(\mathbb{V}))$. The probability that the elements of X is distinct from the marginal probability of a cluster in that $P(X)$ sums the marginal probabilities of all clusters that include all elements of X. Once the marginal probability of each cluster is computed, the marginal probability of any sets of elements being clustered together can be computed in time and space linear in the size of the trellis.

3.1.4 Maximal Cluster Splits and Joins

The trellis can facilitate computing the most likely *join*, i.e., the pair of clusters that maximally increases the resultant clustering potential when combined:

$$\underset{C,C' \in \mathcal{C}}{\operatorname{argmax}} \{\mathcal{E}(C \cup C') * \mathbb{1}_{\{\mathcal{E}(C \cup C') > \mathcal{E}(C) * \mathcal{E}(C')\}}\}.$$

We also consider the most likely *split* of a cluster. For a cluster C, this reduces to finding the MAP estimate for the partitioning of C. This can be computed directly using the subtrellis rooted at the vertex corresponding to C. An alternative approach to using the trellis for finding likely splits for clusters in a clustering is to choose to place a restriction on the set of possible splits, for example that only a single datapoint can be split from an existing cluster $C \in \mathcal{C}$. These splits are the set of children of the verticies in the trellis corresponding to clusters in $\mathcal{C}$, and the one among them that maximally increases the resultant clustering potential is

$$\underset{\mathbb{V}' \in ch(\mathbb{V}(C))}{\operatorname{argmax}} \{\mathcal{E}(X(\mathbb{V}')) * \mathbb{1}_{\{\mathcal{E}(X(\mathbb{V}')) > \mathcal{E}(X(\mathbb{V}))\}} | C \in \mathcal{C}\}$$

where

$$ch(\mathbb{V}) = \{\mathbb{V}' \in \mathbb{V}(\mathbb{T}) | X(\mathbb{V}') \subset X(\mathbb{V}) \wedge ||X(\mathbb{V}')| - |X(\mathbb{V})|| = 1\}$$

and $\mathbb{V}(C)$ is the vertex in $\mathbb{T}$ corresponding to C.

3.1.5 Trellis Construction

For completeness, we present the following algorithm for building a trellis.

Algorithm 3 ConstructTrellis(X)

$\mathbb{V}(\mathbb{T}) \leftarrow \emptyset$
$E(\mathbb{T}) \leftarrow \emptyset$
for $i \leftarrow 1$ through $|\mathbb{P}(X)|$ **do**
  for C in $\mathbb{P}_i(\mathbb{V})$ **do**
    Let $\mathbb{V}$ be a vertex corresponding to C
    $\mathbb{V}(\mathbb{T}) \leftarrow \mathbb{V}(\mathbb{T}) \cup \{\mathbb{V}\}$
    for C' in $\mathbb{P}_{i-1}(C)$ **do**
      Let $\mathbb{V}'$ be the vertex corresponding to C'
      $E(\mathbb{T}) \leftarrow E(\mathbb{T}) \cup \{(\mathbb{V}', \mathbb{V})\}$
    $\mathcal{E}(\mathbb{V}) \leftarrow$ `ComputePotential`$(\mathbb{V})$
return $\mathbb{T}$

3.2 Case Study: Correlation Clustering

The energy-based clustering framework is compatible with any objective computed from a set of non-negative cluster scores[1].

One such objective that is widely used in practice is known as correlation clustering [5]. We present the traditional correlation clustering model in this section and present it in the energy-based correlation clustering model in the next section. The input to correlation clustering is a complete (weighted) graph, $G = (V, E)$, where each edge has real-valued weight, i.e., $w_{uv} \in \mathbb{R},\ (u, v) \in E$. The goal is to construct a clustering of the vertices that maximizes the sum of positive edge-weights within each cluster minus the negative edge-weights across the clusters.

Formally, let $\mathbb{C}_V$ be the set of all clusterings of V. Given a clustering $\mathcal{C} \in \mathbb{C}_V$, the sum of all positive within cluster edge-weights with respect to a clustering $\mathcal{C}$ is denoted $S^+(\mathcal{C}) = \sum_{C \in \mathcal{C}} \sum_{(u,v) \in C} w_{uv} \mathbb{1}_{\{w_{uv} > 0\}}$.

[1]One approach to using the energy-based clustering framework with negative cluster scores is to exponentiate the cluster scores prior to inputing them into the framework, as we will see in this section.

Similarly, $S^-(\mathcal{C})$ is the sum of the negative across-cluster edges with respect to $\mathcal{C}$. The optimal clustering $\mathcal{C}^\star \in \mathbb{C}_V$ is the one that maximizes the sum of positive within-cluster edge weights minus the sum of all negative across-cluster edge weights, $\mathcal{C}^\star = \max_{\mathcal{C}\in\mathbb{C}_V} S^+(\mathcal{C}) - S^-(\mathcal{C}) = \max_{\mathcal{C}\in\mathbb{C}_V} S^\pm(\mathcal{C})$. The problem is known to be NP-Hard[5].

There exist other objective functions over clusterings that are ordering-equivalent to $S^\pm(\cdot)$. Define $S(\mathcal{C}) = \sum_{C\in\mathcal{C}} \sum_{(u,v)\in C} w_{uv}$.

Proposition 5. *Let $\mathcal{O}^\star_{S^\pm}(\mathbb{C}_X)$ be the sequence containing all clusterings of a set of elements, X, in descending order with respect to $S^\pm(\cdot)$. Let $\mathcal{O}^\star_S(\mathbb{C}_X)$ be the sequence containing all clusterings of X, in descending order with respect to $S(\cdot)$. Then $\mathcal{O}^\star_{S^\pm}(\mathbb{C}_X)$ and $\mathcal{O}^\star_S(\mathbb{C}_X)$ are ordering-equivalent.*

Proof. Two functions, f_1,f_2 are said to be order equivalent iff $\forall c_i, c_j \in C, f_1(c_i) <= f_1(c_j) \implies f_2(c_i) <= f_2(c_j)$.

$$
\begin{aligned}
f(c) := &\sum_{c_i\in C} \sum_{u,v\in C_i} w_{uv} * \mathbb{1}_{\{w_{uv}>0\}} \\
&- \sum_{c_i,c_j\in C} \sum_{u\in c_i} \sum_{v\in c_j} w_{uv} * \mathbb{1}_{\{w_{uv}<=0\}} \\
g(c) := &\sum_{c_i\in C} \sum_{u,v\in C_i} w_{uv}
\end{aligned}
$$

We want to show that $f(c)$ is order equivalent with $g(c)$.

Note that

$$
\begin{aligned}
\forall C \in \mathbb{C}_n, &\sum_{c_i,c_j\in C} \sum_{u\in c_i} \sum_{v\in c_j} w_{uv} * \mathbb{1}_{\{w_{uv}<=0\}} \\
&+ \sum_{c_i\in C} \sum_{u,v\in C_i} w_{uv} * \mathbb{1}_{\{w_{uv}<=0\}} = E^-
\end{aligned}
$$

where E^- is a constant function of the input affinity matrix.

$$
\begin{aligned}
g(c) &= \sum_{c_i \in C} \sum_{u,v \in C_i} w_{uv} * \mathbb{1}_{\{w_{uv} > 0\}} \\
&\quad + \sum_{c_i \in C} \sum_{u,v \in C_i} w_{uv} * \mathbb{1}_{\{w_{uv} <= 0\}} \\
g(c) - \sum_{c_i \in C} \sum_{u,v \in C_i} w_{uv} * \mathbb{1}_{\{w_{uv} <= 0\}} &= f(c) \\
&\quad + \sum_{c_i, c_j \in C} \sum_{u \in c_i} \sum_{v \in c_j} w_{uv} * \mathbb{1}_{\{w_{uv} <= 0\}} \\
g(c) &= f(c) + \sum_{c_i, c_j \in C} \sum_{u \in c_i} \sum_{v \in c_j} w_{uv} * \mathbb{1}_{\{w_{uv} <= 0\}} \\
&\quad + \sum_{c_i \in C} \sum_{u,v \in C_i} w_{uv} * \mathbb{1}_{\{w_{uv} <= 0\}} \\
g(c) &= f(c) + E^-
\end{aligned}
$$

□

Proposition 5 is not widely known, though it has occasionally been used implicitly. For example, Kappes et al [38], state $S(\cdot)$ as the correlation clustering objective.

Although the two methods for scoring a correlation clustering (i.e., $S^{\pm}(\cdot)$ vs. $S(\cdot)$) may compute different scores for the same clustering, Fact 5 implies that any clustering $\mathcal{C}$ of a dataset, X, has the same ordering under both objectives. Importantly, the best clustering, $\mathcal{C}^\star$, is equivalent under either objective. Our analysis focuses on $S(\cdot)$ since it is more convenient computationally when the number of clusters is large, which is common in practice [78].

The correlation clustering objective is computed in terms of positive and negative edge weights whereas our framework operates over non-negative potentials. We can use a Gibbs distribution to transform cluster scores to potentials, similar to [38]. Specifically, $\mathcal{E}(\mathcal{C}) = \prod_{C \in \mathcal{C}} \mathcal{E}(C) = \prod_{C \in \mathcal{C}} \exp[\sum_{(u,v) \in C} w_{uv}]$. After computing cluster

potentials, the full probability distribution over clusterings is constructed using the equations in Definition 2.

Computing Cluster Potential

Computing the potential of cluster C requires summing the $\frac{|C|(|C|-1)}{2}$ within-cluster edge weights. Since the number of potential clusters is $2^N - 1$ (for a dataset of size N), the naïve approach sums $\sum_{k=1}^{N} \binom{N}{k} \cdot \frac{|k|(|k|-1)}{2} = 2^{N-3}(N^2 - N) = O(N^2 \cdot 2^N)$ edge-weights.

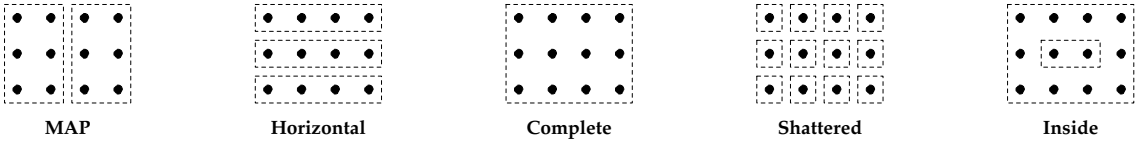
Proposition 6. *Let C be a cluster with $|C| > 2$ and let C_i and C_j be two distinct clusters such that $C_i \subset C, C_j \subset C$ and $|C_i| = |C_j| = |C| - 1$. Then, the potential of C can be expressed as $\mathcal{E}(C) = \frac{\mathcal{E}(C_i)\mathcal{E}(C_j)\mathcal{E}(C_i \setminus C_j \cup C_j \setminus C_i)}{\mathcal{E}(C_i \cap C_j)}$.*

Proposition 6 follows from set algebra and the linearity of the potential function. Algorithm 3 exploits Fact 6 to speed up trellis construction. Algorithm 3 can be found in Section 3.1.5.

Proposition 7. *Algorithm 3 constructs a trellis, $\mathbb{T}$, for a graph $G = (V_G, E_G)$ and computes the potential of all clusters. Computing the potential of all clusters requires $\mathcal{O}(|V(\mathbb{T})|) = \binom{N}{2} + \sum_{k=3}^{N} \binom{N}{k} * 4$, steps where $N = |V_G|$.*

Specifically, cluster potentials are memoized at each vertex in the trellis and then reused to compute the potentials of new clusters before they are added to the trellis. The potential for cluster C corresponding to vertex v in the trellis is denoted $\mathcal{E}(v)$ in Algorithm 3. As described below, `ComputePotential` uses the fast computation described in Fact 6 with memoized values for each of the $\mathcal{E}$ terms at corresponding vertices in the trellis.

(a) `Grid` dataset.



(b) Clustering probabilities.

Data	Clustering	Prob.	Potential
Grid	MAP	2.262e-06	10.206
	Horizontal	5.403e-07	2.437
	Complete	2.216e-07	1.000
	Shattered	2.216e-07	1.000
	Inside	7.968e-08	0.356

Figure 3.1: Probability of clusterings of the `Grid` dataset.

3.3 Experiments

3.3.1 Synthetic Data Example

We begin by providing a simple synthetic data example. We provide the probabilities and potentials of various clusterings of a `Grid` dataset, in which potentials are computed by correlation clustering and exponentiating the negative Euclidean distance between examples (which are simply evenly spaced points on a grid). Notice that the MAP clustering and other clusterings in the Grid dataset exhibit relatively similar probabilities.

3.3.2 Cancer Gene Expression

In this section, we demonstrate the utility of the cluster trellis for flat clustering via experiments on real-world gene expression data. To begin, we provide a high-level background on cancer subtypes to motivate the use of our method in the experiment in Section 3.3.2.

Background For an oncologist, determining a prognosis and constructing a treatment plan for a patient is dependent on the subtype of that patient's cancer [45]. This is because different subtypes react well to some treatments, for example, to radiation and not chemotherapy, and for other subtypes the reverse is true [65]. For example, basal and erbB2+ subtypes of breast cancer are more sensitive to paclitaxel- and doxorubicin-containing preoperative chemotherapy (approx. 45% pathologic complete response) than the luminal and normal-like cancers (approx. 6% pathologic complete response)[61]. Unfortunately, identifying cancer subtypes is often non-trivial. One common method of learning about a patient's cancer subtype is to cluster their gene expression data along with other available expression data for which previous treatments and treatment outcomes are known [66].

Data & Methods We use breast cancer transcriptome profiling (FPKM-UQ) data from The Cancer Genome Atlas (TCGA) because much is known about the gene expression patterns of this cancer type, yet there is heterogeneity in the clinical response of patients who are classified into the same subtype by standard approaches [77].

The data are subselected for African American women with Stage I breast cancer. We select African American women because there is a higher prevalence of the basal-like subtype among premenopausal African American women [55] and there is some evidence that there is heterogeneity (multiple clusters) even within this subtype [77]. Stage I breast cancer patients were selected because of the prognostic value in distinguishing aggressive subtypes from non-aggressive subtypes at an early stage.

Despite the considerable size of TCGA, there are only 11 samples meeting this basic, necessary inclusion/exclusion criteria. Each of the 11 samples is a 20,000 dimensional feature vector, where each dimension is a measure of how strongly a given gene is expressed. We begin by sub-selecting the 3000 features with greatest variance across the samples.

We then add an infinitesimal value prior to taking the log of the remaining features, since genome expression data is believed to be normally distributed in log-space [63]. A similar data processing was shown to be effective in prior work [63].

We use correlation clustering as the energy model. Pairwise similarities are exponentiated negative euclidean distances.

We subtract from each the mean pairwise similarity so that similarities are both positive and negative. We then compute the marginal probabilities for each pair (i.e., the probability that the two samples appear in the same cluster). See Section 3.1.3 for how to compute these values using the trellis.

Model Evaluation using Marginals One method for evaluating a set of cancer subtype clustering models is to identify pairs of samples that the evaluator believes should be clustered together and inspect their pairwise potentials. However, high pairwise potentials do not necessarily mean the points will be clustered together by the model (which considers how the pairs' cluster assignment impacts the rest of the data). Similarly, a low pairwise potential does not necessarily mean the two samples will not be clustered together. The pairwise marginal on the other hand exactly captures the probability that the model will place the two samples in the same cluster. We test if the corresponding unnormalized pairwise potentials or a simple approximation of the marginals could reasonably be used as a proxy for exact pairwise marginals.

Pairwise potentials vs. Marginals & Exact vs. Approximate Marginals Figure 3.2 plots the pairwise log potentials vs. pairwise log marginals of the subsampled TCGA data[2]. The pairwise scores and marginals are not strongly correlated, which suggests that unnormalized pairwise potentials cannot reasonably be used as a proxy for pairwise marginals. For example, the sample pair of patients (partial id numbers given) `74ca` and `d6fa` have a potential of -4.7 (low), but a pairwise marginal

[2]The MAP clustering of the TCGA subsample is:
C1: {`7b57`, `74ca`, `28a2`, `73ac`, `200e`, `62da`, `d6fa`, `c532`}, **C2**: {`6a88`, `0232`}, **C3**: {`a8f5`}.

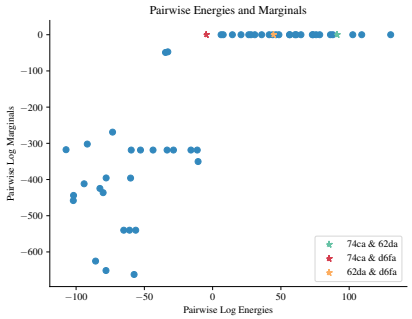


Figure 3.2: For each pair of patients with Stage I cancer, we plot the potential and marginal probability of the pair being in the same cluster as described in Section 3.3.2.

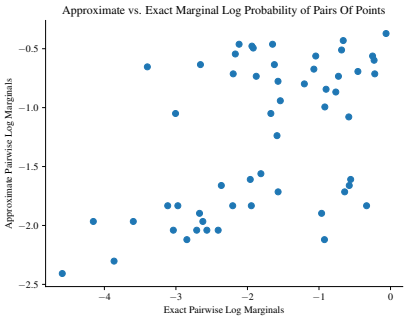


Figure 3.3: The approximate vs. exact pairwise marginals for each pair of gene expressions. Approximate marginals are computed using a Perturb-and-MAP based method [38].

that is nearly one. This is because both `74ca` and `d6fa` have high potential with sample `62da`, with pairwise potentials 91.09 (the fourth largest) and 44.5, respectively. Figures 3.4 and 3.5 that visualize the pairwise potentials and pairwise marginals, respectively.

We also explore the extent to which an approximate method can accurately capture pairwise marginals. We use an approach similar to Perturb-and-MAP [38]. We sample clusterings by adding Gumbel distributed noise to the pairwise potentials and using Algorithm 2 to find the maximal clustering with the modified potentials. We approximate the marginal probability of a given pair being clustered together by measuring how many of these sampled clusters contain the pair in the same cluster.

Figure 3.3 plots the approximate vs. exact pairwise marginal for each pair of points in the dataset. The figure shows that the approximate method overestimates many of the pairwise marginals. Like the pairwise scores (rather than exact marginals), using the approximate marginals in practice may lead to errors in data analysis.

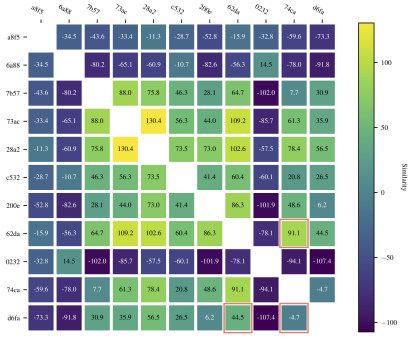


Figure 3.4: Heatmap of the pairwise potentials between the patients. The pair `74ca` and `d6fa` has a potential of -4.7, `74ca` and `62da` have 91.09, and `d6fa` and `62da` have 44.5.

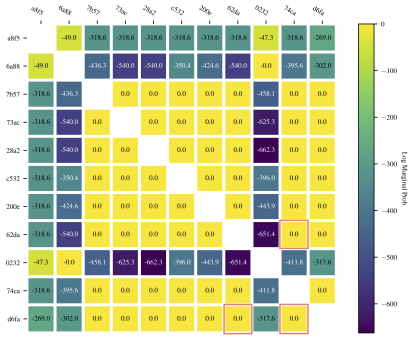


Figure 3.5: Heatmap of the marginal probability that a pair will be clustered together. Patients `74ca` and `d6fa` have a pairwise marginal that is nearly one, despite having a low pairwise potential.

3.3.3 Pairwise Potentials vs. Marginals in UCI Zoo Dataset

We repeat our experiment comparing pairwise potentials vs. marginals, as described in section 4.2, using data selected from the UCI zoo dataset [28]. Potentials are computed by exponentiated cosine similarity. Figures 3.6 and 3.7 show that the potentials and marginals for many pairs are not well correlated. For example, the pair sea wasp and termite has high potential, however the marginal probability of the pair being clustered together is low.

Figure 3.6: Heatmap of the pairwise potentials between the animals.

Figure 3.7: Heatmap of the marginal probability that a pair of animals will be clustered together.

3.4 Related Work

While there is, to the best of our knowledge, no prior work on compact representations for exactly computing distributions over clusterings, there is a small amount related work on computing the MAP k-clustering exactly, as well as a wide array of related work in approximate methods, graphical models, probabilistic models for clustering, and clustering methods.

Previous work explores using trees to encode distributions over clusterings, though the focus is limited to modeling mixtures of tree consistent partitions rather than computing the marginals, maximal clusterings, and the partition function [33, 8].

The first dynamic programming approach to computing the MAP k-clustering was given in [36], which focuses on minimizing the sum of square distances within clusters. It works by considering distributional form of the clusterings, i.e., all possible sizes of the clusters that comprise the clustering, and defines "arcs" between them. However, no full specification of the dynamic program is given and, as the author notes, many redundant computations are required, since there are many clusterings that share the same distributional form. In [35], the first implementation is given, with some of the redundancies removed, and the implementation and amount of redundancy is further improved upon in [71]. In each of these cases, the focus is on finding the best *k-clustering*, which can be done using these methods in $\mathcal{O}(3^n)$ time. These methods can also be used to find the MAP clustering for all K, however doing so would result in an $\mathcal{O}(n * 3^n)$ time, which is worse than our $\mathcal{O}(3^n)$ result.

In [42], the authors use fast convolutions to compute the MAP k-clustering and k-partition function. Fast convolutions use a Mobius transform and Mobius inversion on the subset lattice to compute the convolution in $\widetilde{\mathcal{O}}(n^2 2^n)$ time. It would seem promising to use this directly in our work, however, our algorithm divides the subset lattice in half,which prevents us from applying the fast transform directly. The authors note that, similar to the above dynamic programming approaches, their method can be used to compute the clustering partition function and MAP in $\mathcal{O}(n * 3^n)$, which is larger than our result of $\mathcal{O}(3^n)$. Their use of convolutions to compute posteriors of k-clusterings also implies the existence of an $\widetilde{\mathcal{O}}(n^3 2^n)$ algorithm to compute the pair-wise posterior matrix, i.e., the probability that items i and j are clustered together, though the authors mention that, due to numerical instability issues, using fast convolutions to computing the pair-wise posterior matrix is only faster in theory.

Recently proposed perturbation based methods [38] approximate distributions over clusterings as well as marginal distributions over clusters. They use the Perturb and MAP approach [57], originally proposed by Papandreou, which is based on adding

Gumbel distributed noise to the clustering potential function. Unfortunately, for Perturb and MAP to approach the exact distribution, independent samples from the Gumbel distribution must be added to each clustering potential, which would require a super-exponential number of draws. To overcome this, Kappes et al. [38] propose adding Gumbel noise to the pairwise real-valued affinity scores, thus requiring fewer draws, but introducing some dependence among samples. They must also perform an outer relaxation in order obtain a computable bound for the log partition function. As a result, the method approaches a distribution with unknown approximation bounds.

CHAPTER 4

APPROXIMATING DISTRIBUTIONS OVER FLAT CLUSTERINGS

While exact methods can be used for relatively small data sets, we wish to be able to compute distributions over clusterings for larger, less-constrained data sets as well. In this chapter we describe approaches to approximating distributions over clusterings for larger data sets than can be handled with exact methods, by utilizing sparse trellis or slab tree data structures.

4.1 Algorithms and Analysis

4.1.1 Sparse Trellises

For trellises closed under recursive complement or those that are tree-structured, we show that the partition function, the clustering with maximal potential, and marginal probabilities (of a given cluster, or of a given set of points being in the same cluster) can be computed using the algorithms described in Section 3.1. When the vertices omitted from a sparse trellis correspond to clusters that have non-zero potential, the algorithms provide approximate rather than exact values for the clustering model. Since these algorithms have complexity measured in the number of nodes in the trellis, their efficiency improves with trellis-sparsity. The more general family of all sparse trellises is also discussed briefly.

4.1.1.1 Approximating the Partition Function

Sparse Trellises Closed Under Recursive Complement

Recall from definition 9 that a sparse trellis is closed under recursive complement if and only if for every ancestor of a vertex in the trellis there exists a complementary vertex, such that the dataset associated with the vertex and the dataset associated with the complementary vertex have empty intersection and union equal to the dataset associated with the ancestor. It important to note that if a $\widehat{\mathbb{T}}$ is not closed under recursive complement, we cannot simply run Algorithm 1 because not all vertices for which the algorithm must compute the potential (or the partition function) are guaranteed to exist.

Given a sparse trellis, $\widehat{\mathbb{T}}$, that is closed under recursive complement, we are able to compute the partition function, i.e., the sum of the potentials over of all sparse-trellis-consistent clusterings, $Z(\widehat{\mathbb{T}}) = \sum_{\mathcal{C} \in \mathbb{C}(\widehat{\mathbb{T}})} \mathcal{E}(\mathcal{C})$, by using Algorithm 1.

Proposition 8. *Let $\widehat{\mathbb{T}} = (\mathbb{V}(\widehat{\mathbb{T}}), E(\widehat{\mathbb{T}}))$ be a sparse trellis whose vertices are closed under recursive complement. Then Algorithm 1 computes $Z(\widehat{\mathbb{T}})$ in $\mathcal{O}(|\widehat{\mathbb{T}}|^{log(3)})$.*

Proof. Recall that Algorithm 1 begins by constructing $\mathbb{V}(\mathbb{T})^{(i)}$ with respect to an arbitrary element x_i. Analogously, for the sparse trellis $\widehat{\mathbb{T}}$ we construct $\widehat{\mathbb{V}}^{(i)}$. Note that $|\widehat{\mathbb{V}}^{(i)}| < 2^{N-1}$ or else $\widehat{\mathbb{T}} = \mathbb{T}$. By our zero-potential assumption, for all $\mathbb{V} \in \mathbb{V}(\mathbb{T})^{(i)} \backslash \widehat{\mathbb{V}}^{(i)}$, $\mathcal{E}(\mathbb{V}) = 0$. Therefore, the potential of any clustering that is not computed by Algorithm 1 is also zero and may be omitted when computing the partition function of $\widehat{\mathbb{T}}$.

Finally, we show by contradiction that the algorithm does not omit any clusterings with non-zero potential. Assume that the algorithm does not compute the potential of a clustering $\mathcal{C}$ that has non-zero potential. Since $\mathcal{C}$ is a valid clustering, it must contain a cluster C that contains the element x_i. If $C \in \widehat{\mathbb{T}}$ then the vertex $\mathbb{V}$ that represents C must be in $\widehat{\mathbb{V}}^{(i)}$ and $\mathcal{E}(\mathbb{V}) > 0$, so the algorithm would have computed its potential. Therefore, $\mathbb{V} \notin \widehat{\mathbb{V}}^{(i)}$ which means that $\mathcal{E}(C) = 0$, a contradiction.

By Theorem 1, this algorithm runs in $\mathcal{O}(|\widehat{\mathbb{T}}|^{log(3)})$ time and space linear in $|\widehat{\mathbb{T}}|$. Note that due to the constraint that $\widehat{\mathbb{T}}$ be closed under recursive complement, for every vertex $\mathbb{V}$ in the sparse trellis such that $\mathbb{V}$ has no parents in $\widehat{\mathbb{T}}$, each element $x_i \in X(\mathbb{V})$ is contained in $|\widehat{\mathbb{T}}[\mathbb{V}]|/2$ clusters, allowing the same counting argument as in the proof of Theorem 1. □

Tree-structured Sparse Trellises

As mentioned in Section 2.2.2, the often-used hierarchical (tree structured) clustering encompasses one family of sparse trellises.

We can use a similar technique to compute the partition function of a tree-structured trellis.

Fact 1. *Let p be a parent vertex and let* `ch(p)` *be p's children. Beginning at the leaves, proceed up the tree computing $Z(p) = \mathcal{E}(p) + \prod_{c \in \texttt{ch(p)}} Z(c)$, where $Z(\cdot)$ is the (memoized) partition function at a node. Then $Z(root)$ will contain the partition function for tree-consistent partitions.*

Proof. We must compute the partition at p. Note that if p is a leaf, the partition function at p is $\mathcal{E}(p)$. Otherwise, let `ch(`p`)` be the children of p. First, note that a valid clustering of $X(p)$ consist of the union of a clustering from each of the children of p, $c \in$ `ch(`p`)`. The same is true for c, and all of p's descendents. Also, note that the potential of this sampled clustering is simply the product of the potentials of the samples. Recall that a partition function of a child $\mathbb{V} \in$ `ch(p)`, $Z(\mathbb{V})$ is a sum of clustering potentials over all clusterings of the descendants of $\mathbb{V}$. Therefore, $\prod_{\mathbb{V}' \in \texttt{ch(p)}} Z(\mathbb{V}')$ is a sum of terms, each term being a product of one clustering from each child in `ch(p)`, i.e., a valid clustering of the descendants of p. Notice that this product contains *all* valid clusterings of the descendants of p such that each valid clustering is built by taking the union of a clustering from each child in `ch(p)`. Also, no clustering will be double counted because all of p's children are disjoint. Adding

the potential of the complete partition, $\mathcal{E}(p)$, to this product computes the partition function of p. $\square$

4.1.1.2 Approximating the MAP Clustering

Sparse Trellises Closed Under Recursive Complement

Given a sparse trellis closed under recursive complement, $\widehat{\mathbb{T}}$, closed under recursive complement, we are able to compute the MAP sparse-trellis-consistent partition, $\mathcal{C}^{\star}(\widehat{\mathbb{T}}) = argmax_{\mathcal{C}\in\mathbb{C}(\widehat{\mathbb{T}})}\mathcal{E}(\mathcal{C})$, by using Algorithm 2. Doing so takes $\mathcal{O}(|\widehat{\mathbb{T}}|^{log(3)})$ time and $\mathcal{O}(|\widehat{\mathbb{T}}|)$ space. The correctness and complexity analysis are analogous to those given for Proposition 8, with references to the partition function replaced with MAP clustering and references to Theorem 1 with Proposition 4.

Tree-structured Sparse Trellises

As with the parititon function, we are able to compute the maximal clustering in a tree-structured sparse trellis, $\mathbb{T}$, in $\mathcal{O}(|\widehat{\mathbb{T}}|)$ time an a similar manner. Namely, starting at the leaves, compare the potentials of a parent and the product of its childrens' potentials. Store the maximum of these two options at the parent, along with the corresponding clustering (either the parent or the union of the clusterings stored at each child). Continue the process upwards until the root of the trellis is reached. At the end of this process the root contains the clustering with the maximal potentials as well as the corresponding potentials. The proof of correctness is analogous to the one given for Fact 1.

4.1.1.3 Approximating Marginals

The sparse trellis facilitates the computation of the same two types of cluster marginals as the full trellis: (1) the probability of a specific cluster, X_i, with respect to the distribution over all possible clusterings, and (2) the probability that any group of elements, X_i, are clustered together.

The probability of cluster X_i, given sparse trellis $\widehat{\mathbb{T}}$, denoted $P_{\widehat{\mathbb{T}}}(\mathbb{V}(X_i))$, is 0 if $X_i \notin \mathbb{V}(\widehat{\mathbb{T}})$, i.e., if data set X_i is not in sparse trellis $\widehat{\mathbb{T}}$. This is not necessarily the case for the probability that a group of elements, X_i, are clustered together.

Sparse Trellises Closed Under Recursive Complement Given a sparse trellis, $\widehat{\mathbb{T}}$, that is closed under recursive complement, and a cluster, $X(\mathbb{V})$, s.t. $\mathbb{V} \in \mathbb{V}(\widehat{\mathbb{T}})$, the algorithm described in Section 3.1.3 can be used to compute the probability $X(\mathbb{V})$ by modifying the set of clusterings being summed over–from all possible clusterings to those that are consistent with the sparse trellis and containing the data set $X(\mathbb{V})$, denoted $\mathbb{C}(\widehat{\mathbb{T}})^{(\mathbb{V})}$:

$$P(X(\mathbb{V})) = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\widehat{\mathbb{T}})^{(\mathbb{V})}}} \mathcal{E}(\mathcal{C})}{Z(X)}$$

where $Z(X)$ is the partition function with respect to the sparse trellis, described in Section 4.1.1.1. Re-writing this probability in terms of the complement of $X(\mathbb{V})$, we get,

$$P(X(\mathbb{V})) = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\widehat{\mathbb{T}})^{(\mathbb{V})}}} \mathcal{E}(\mathcal{C})}{Z(X)} = \frac{\sum_{\mathcal{C} \in \mathbb{C}^{(\widehat{\mathbb{T}})^{(\mathbb{V})}}} \mathcal{E}(X(\mathbb{V}))\mathcal{E}(\mathcal{C} \setminus X(\mathbb{V}))}{Z(X)} \tag{4.1}$$

$$= \frac{\mathcal{E}(X(\mathbb{V})) \sum_{\mathcal{C}' \in \mathbb{C}(\widehat{\mathbb{T}})_{X \setminus X(\mathbb{V})}} \mathcal{E}(\mathcal{C}')}{Z(X)} = \frac{\mathcal{E}(X(\mathbb{V}))Z(X \setminus X(\mathbb{V}))}{Z(X)} \tag{4.2}$$

As in the full trellis case, additional Z values may need to be computed, and the complexity of computing the partition function of the remaining vertices is no greater than the complexity of Algorithm 1 on a sparse trellis, namely $\mathcal{O}(|\widehat{\mathbb{T}}|^{log(3)})$.

Given a set of elements, X, the marginal probability of the elements of X being clustered together is: $P(X) = \sum_{X(\mathbb{V}) \in \widehat{\mathbb{T}}: X \subseteq X(\mathbb{V})} P(X(\mathbb{V}))$. Once the marginal probability of each cluster in the sparse trellis is computed, the marginal probability of any sets of elements being clustered together can be computed in time and space linear in the size of the sparse trellis.

Tree-Structured Sparse Trellises Given a tree-structured sparse trellis, $\widehat{\mathbb{T}}$, and a cluster $X_i \in \widehat{\mathbb{T}}$, the marginal probability of cluster X_i, $P(\mathbb{V}(X_i))$, is given by the product of the cluster potential with the product of all siblings and aunt vertices in the tree, i.e.,

$$P(\mathbb{V}(X_i)) = \mathcal{E}(X_i) * \prod_{\mathbb{V}_j \in \mathbb{V}(\mathbb{T}) | X_i \subseteq X(\mathbb{V}_j)} Z(X(\mathbb{V}_j) \setminus X_i) \tag{4.3}$$

The marginal probability of a set of elements, X_i, being clustered together is computed by summing the marginal probabilities of all clusters in the sparse trellis that contain X_i, i.e.,

$$P(X_i) = \sum_{\mathbb{V}_j \in \mathbb{V}(\mathbb{T}) | X_i \subseteq X(\mathbb{V}_j)} P(\mathbb{V}_j) \tag{4.4}$$

The proofs are left as an exercises for the interested reader.

4.1.1.4 Sparse Trellis Construction

There are two general approaches to building sparse trellises: sparsification (subtractive), and construction (additive) methods. While sparsification methods are interesting from a theoretical perspective, for example one can consider compression rates when removing vertices from a trellis, any method that starts with a full trellis will necessarily not scale. Therefore, we focus on methods for sparse trellis construction.

Inspired by Hierarchical Agglomerative Clustering (HAC), we propose a sparse trellis building method DAG Agglomerative Clustering (DAC). DAC works in a manner similar to HAC, only in DAC, when two clusters are merged, they are not removed from the queue, as in HAC, i.e., skip line 9 in the HAC pseudocode given in [1]. Instead of stopping when a tree is reached, DAC stops when some specified criteria are met, for example the total number of vertices added or the relative potential of recently added vertices.

Sparse trellises built using DAC might not be closed under recursive complement. In such a case, we are able to close a sparse trellis by adding vertices (and edges) to it, while setting the potential for each vertex as 0. The majority of sparse trellises are not closed under recursive complement, therefore when given an arbitrarily-shaped sparse trellis, it may be necessary to add a large number of vertices, impacting the time and space complexity of using trellises that are built and then subsequently closed under recursive complement. See Section 7.3 for more on building sparse trellises.

We note that there also exist numerous algorithms for building tree structures, which can be utilized directly to create tree-structured sparse trellises (HAC being one such algorithm).

4.1.2 Slab Trees

Recall that slab trees distribute potential to clusterings that are not tree consistent and otherwise would all be given zero probability mass. Our goal is to efficiently approximate distributions over clusterings of a set of points in a manner that 1) is able to assign non-zero probability mass to every clustering, 2) minimizes divergence to the true distribution over clusterings, and 3) forms a true probability distribution (i.e., sums to 1), and that . We begin by describing an algorithm for assigning potential to clusterings that allows non-zero probability mass for clusterings that are not tree consistent while still forming a valid distribution. We then describe how to set a slab tree's node potential and slab values in order to minimizes total variation distance between the approximate and exact distributions. We compare our approach to an approximate normalization method, which sets the potential for each clustering, $\mathcal{C}$, as given in Definition 2, and uses an approximation of the partition function to produce probabilities (see Equation 4.7), as is done, e.g., in [38].

4.1.2.1 Assigning Clustering Potentials

In order to assign potential to a given clustering, $\mathcal{C}$, using a slab tree, $\mathbb{T}$, we define a function, $f_{\mathbb{T}} : \mathbb{C} \rightarrow \mathbb{P}(\mathbb{V}(\mathbb{T}))$, over the set of clusterings and slab trees that maps clusterings to a subset of vertices in the slab tree.

$$f_{\mathbb{T}}(\mathcal{C}) = \{\mathbb{V} | \mathbb{V} \in \mathbb{V}(\mathbb{T}) \wedge (\forall C \in \mathcal{C}, C \subseteq X(\mathbb{V}) \vee C \bigcap X(\mathbb{V}) = \emptyset)\} \wedge$$
$$\forall \mathbb{V}' \in \{\mathbb{V} \in \mathbb{V}(\mathbb{T}) | \forall C \in \mathcal{C}, C \subseteq X(\mathbb{V}) \vee C \bigcap X(\mathbb{V}) = \emptyset\}, \mathbb{V} \subseteq \mathbb{V}'\}$$

In words, f maps clustering $\mathcal{C}$ to the set of vertices in $\mathbb{T}$ s.t. (1) all the clusters in $\mathcal{C}$ are either a subset of data set corresponding to each vertex in the set or has empty intersection each such data set, and (2) no vertex in the set is a proper superset of another vertex in the set. The potential for $\mathcal{C}$ is then given by

$$\mathcal{E}(\mathcal{C}) = \prod_{\mathbb{V} \in f_{\mathbb{T}}(\mathcal{C})} \mathcal{E}(\mathbb{V}) \times \mathbb{1}_{\{X(\mathbb{V}) \in \mathcal{C}\}} + \mathcal{S}(\mathbb{V}) \times \mathbb{1}_{\{X(\mathbb{V}) \notin \mathcal{C}\}} \tag{4.5}$$

See Algorithm 4 for pseudo-code to compute $\mathcal{E}(\mathcal{C})$ using the root of slab tree $\mathbb{T}$, $\mathbb{V}$(X). Note that since the potential for each cluster in a clustering must be computed and it takes $\mathcal{O}(log(N))$ to determine the cluster potential using a slab tree, thus the runtime complexity of is assigning the potential of a clustering is $\mathcal{O}(Nlog(N))$. Given this approach to assigning the potentials of clusterings, we are able to compute the partition function using the following algorithm:

$$Z(\mathbb{V}) = \mathcal{E}(\mathbb{V}) + \mathcal{S}(\mathbb{V})(B_{|X(\mathbb{V})|} - \prod_{\mathbb{V}' \in \mathsf{ch}(\mathbb{V})} B_{|X(\mathbb{V}')|} - 1) + \prod_{\mathbb{V}' \in \mathsf{ch}(\mathbb{V})} Z(\mathbb{V}') \tag{4.6}$$

where B_N is the N^{th} Bell number.

Theorem 2. *Computing $\mathcal{E}(\mathcal{C})$ as in Equation 4.5, and $Z(X) = Z(\mathbb{V}(X))$ as in Equation 4.6 results in a correct partition function, i.e., $\sum_{\mathcal{C} \in \mathbb{C}} \mathcal{E}(\mathcal{C})/Z(\mathbb{V}(X)) = 1$.*

Proof. We use a strong inductive argument. The base cases of $N = 1, 2, 3$ are trivial. The inductive step follows directly from 4.6 and Fact 1. □

Algorithm 4 Compute-Potential($\mathbb{V}, \mathcal{C}$))

```
for C in 𝒞 do
  for 𝕍′ in ch(𝕍) do
    if C ⊄ X(𝕍′) and C ⋂ X(𝕍′) ≠ ∅ then
      if C = X(𝕍) then
        return ℰ(𝕍)
      else
        return 𝒮(𝕍)
return ∏_{𝕍′∈ch(𝕍)} Compute-Potential(𝕍′, 𝒞)
```

4.1.2.2 Setting Vertex Potential and Slab Values

A natural question arises regarding how slab values should be set. Since slabs distribute potential to clusterings in a way that depends on which vertices in the slab tree every cluster in the clustering is consistent with, this may be viewed as quantizing clusterings to the "nearest" tree-consistent clustering, where "nearest"is based on these vertices. Therefore, the best possible value to assign a slab when minimizing the total variation distance is the mean potential of the sub-clusterings that will assigned the slab value. More precisely,

Proposition 9. *The optimal value to assign slab potential for vertex $\mathbb{V}$ in order to minimize total variation distance can be computed as follows:* $\mathcal{S}(\mathbb{V}) = \frac{Z(\mathbb{V}) - \mathcal{E}(\mathbb{V}) - \prod_{\mathbb{V}' \in ch(\mathbb{V})} Z(\mathbb{V}')}{B_{|X(\mathbb{V})|} - \prod_{\mathbb{V}' \in ch(\mathbb{V})} B_{|X(\mathbb{V}')|} - 1}$, *where B_N is the N^{th} Bell number.*

Proof. We use a strong inductive argument. The base cases of $N = 1, 2, 3$ are trivial. The inductive step, note the numerator computes the sum of the potentials of the (sub)clusterings being given the slab value $S(V)$, and the denominator is the number of (sub)clusterings that will be given the slab value $S(V)$, which gives the mean. The mean provides the minimum variance for the slab $S(V)$, which in turn minimizes the

total variation distance, since the total variation distance is a monotonic function of the slab variances. □

A more subtle question is how the potentials at each vertex should be set. One might chose to set them to be the same as the slab value, simplifying equation 4.6 somewhat. Another natural choice is to set the vertex potentials to be the same as would be used in a tree-structured sparse trellis tree shape, which has the property that tree consistent clusterings are given the same potential as they would be in tree-structured sparse trellises. We choose to use the latter approach in our experiments.

Note that the setting of optimal slab values utilizes the partition function value at each vertex in the slab tree. When the exact partition function is unavailable, as might be expected, we use an approximate value to set the slab values.

4.1.2.3 Slab Tree Construction

Even with perfect partition function estimates, the total variation distance of slab trees can vary based on the vertices making up the tree. The slab tree that minimizes the total variation distance from the exact distribution is denoted as $\mathbb{T}^\star$. Note that the total variation distance between a slab tree and the exact distribution is a function of both the tree topology as well as the quality of the partition function estimate used to set the slab values. Assuming a perfect partition function estimate, the optimal tree can be expressed as the one that minimizes the variance of the clustering potentials that cross each cut represented by the tree structure of $\mathbb{T}^\star$. In the case where the tree structure represents primarily high potential clusters and the slab value provides potential for the remaining, relatively low-potential clusters, the minimization is similar in form to sparsest cut [4], motivating the use of Sparsest-Cut Trees. Sparsest-Cut Trees have been popular in the Bayesian hierarchical clustering community, since Sparsest-Cut Trees are a greedy approximation to a natural cost function on trees [25]. Sparsest-Cut Trees are built by repeatedly taking the sparsest

cut, starting at the root and ending at the leaves. Since the sparest-cut problem is known to be NP-hard [49], approximations are used in practice. We use Sparsest-Cut Trees as a heuristic for slab tree tree-structure in our experiments in Section 4.2.

4.1.2.4 Slab Tree vs Distributions Using an Approximate Normalizer

An alternative to approximating the distribution over clusterings using a hierarchical clustering, is to approximate the value of the partition function directly. In this approach, the probability of a clustering is defined as:

$$P_X(\mathcal{C}) = \frac{\mathcal{E}_X(\mathcal{C})}{\hat{Z}_X}, \tag{4.7}$$

where $\hat{Z}_X$ is the approximation to the true partition function Z_X. Perturb-and-MAP is an approach for computing $\hat{Z}_X$ that has been applied to energy-based product partition models [38].

An interesting characteristic of computing the probability of $\mathcal{C}$, is that using the exact clustering potential as the numerator and an estimate of Z in the denominator is that the $\|L\|_1$ distance of the resultant distribution is entirely a function of the absolute difference between the estimated and exact partition functions. In particular, the L_1 distance between the true distribution P and the approximate distribution $\hat{P}$ is given by $\left\|P - \hat{P}\right\|_1 = \frac{|\hat{Z}-Z|}{|\hat{Z}|}$.

Using approximate normalizers to estimate probability distributions does *not* result in probability distributions over clusterings. In particular, the sum of clustering probabilities assigned using Equation 4.7 do not equal 1, except in the case where the partition function estimate, $\hat{Z_X}$, equals the exact partition function Z_X. This violates the Kolmogorov axiom of unit measurement[43], one of the three basic axioms of probability theory, thus is undesirable from a theoretical perspective. It may be

undesirable in practice as well[1]. Recall that the slab tree methods satisfy the axiom of unit measurement.

In addition, it is possible to quantify when slab trees provide a smaller total variation distance from the true distribution than the using an approximate normalizer:

Proposition 10. *Compared with approximate normalization methods, distributions approximated with slab trees result in a reduction in total variation distance from the exact distribution when the following holds:*

$$|\hat{Z} - Z| > \sum_{\mathcal{C} \in \mathbb{C}} \left\| \frac{\hat{Z}}{Z} \mathcal{E}(\mathcal{C}) - \hat{\mathcal{E}}(\mathcal{C}) \right\|$$

where $\mathcal{E}_{\mathbb{T}}(\mathcal{C})$ is the potential assigned to the clustering $\mathcal{C}$ by slab tree $\mathbb{T}$.

Proof. This follows from the following algebra:

$$\sum_{\mathcal{C} \in \mathbb{C}} \left\| \frac{\mathcal{E}(\mathcal{C})}{Z} - \frac{\mathcal{E}(\mathcal{C})}{\hat{Z}} \right\| > \sum_{\mathcal{C} \in \mathbb{C}} \left\| \frac{\mathcal{E}(\mathcal{C})}{Z} - \frac{\mathcal{E}_{\mathbb{T}}(\mathcal{C})}{\hat{Z}} \right\|$$

$$\implies \frac{|\hat{Z} - Z|}{|\hat{Z}|} > \sum_{\mathcal{C} \in \mathbb{C}} \left\| \frac{\mathcal{E}(\mathcal{C})}{Z} - \frac{\mathcal{E}_{\mathbb{T}}(\mathcal{C})}{\hat{Z}} \right\|$$

$$\implies Z \cdot |\hat{Z} - Z| > \sum_{\mathcal{C} \in \mathbb{C}} \left\| \hat{Z}\mathcal{E}(\mathcal{C}) - Z\mathcal{E}_{\mathbb{T}}(\mathcal{C}) \right\|$$

$$\implies Z \cdot |\hat{Z} - Z| > Z \sum_{\mathcal{C} \in \mathbb{C}} \left\| \frac{\hat{Z}}{Z} \mathcal{E}(\mathcal{C}) - \mathcal{E}_{\mathbb{T}}(\mathcal{C}) \right\|$$

□

Note that as $|\hat{Z} - Z|$ approaches zero, so does the total variation distance of approximate normalization methods from the exact distribution. Conversely, the slab tree total variation distance from the exact distribution approaches zero as $\mathcal{E}_{\mathbb{T}}(\mathcal{C})$ approaches $\frac{\hat{Z}}{Z}\mathcal{E}(\mathcal{C})$.

---

[1]We, like Kolmogorov, "disregard the deep philosophical dissertations on the concept of probability in the experimental world," and instead refer the reader to [50].

4.2 Experiments

We measure the divergence between the distributions produced using our method and the true distribution over clusterings on small datasets, where doing so is computationally feasible. We propose and analyze an evaluation criteria for this setting. In our experiments, we evaluate using several clustering datasets in applications areas including entity resolution, phylogenetics, and biomedical gene expression, for which distributions over clusterings are particularly useful.

4.2.1 Application Areas & Data

Entity Resolution Knowledge bases (KBs), structured data repositories that express relationships between entities, are ubiquitous in modern technology, for example, in web-search, automated assistants, map services (such as Google Maps), and academic bibliographic services (such as Semantic Scholar or PubMed). A fundamental component of the automatic construction of these knowledge bases is *entity resolution*, the task of determining the identity of ambiguous entity mentions. We apply our method in the entity resolution setting for *scientific author co-reference*. In building a knowledge base of scientists and research papers, such as Google Scholar, author names appearing on scientific papers may be ambiguous. For example, DBLP lists numerous real-world authors who share the name Wei Wang[2]. We apply our method to a recently created dataset of scientific author entity mentions used in [80]. We use pairwise similarities given by the state-of-the-art graph autoencoder based approach [80].

Phylogenetics Tree structures are often used in biology to organize the relationships between organisms at multiple levels of granularity. The distribution over clusterings given by the slab tree model captures additional uncertainty beyond that offered by a

[2] `https://dblp.uni-trier.de/pers/hd/w/Wang:Wei`

traditional hierarchical clustering, providing a richer representation of the evolutionary relationship between biological organisms. We use the popular UCI Zoo dataset[3], which consists of animals and their attributes. We obtain pairwise edge scores by computing the mean-subtracted cosine distance for all pairs of data points. We select a subset of animals from related classes.

Gene Expression in Breast Cancer Patients Oncologists develop treatment plans for patients based on the patients' subtype of cancer [45]. The subtype of cancer is often determined based on clustering patients' gene expression data combined with expression data from previous treatments for which outcomes are known [65]. The Cancer Genome Atlas (TCGA) provides breast cancer transcriptome profiling (FPKM-UQ) data. We select a subset of African American women with Stage I breast cancer for the reasons given in [31]. We use the same pre-processing techniques described in [31].

4.2.2 Methods

We compare the following methods:

Perturb-and-MAP [38] By repeatedly adding Gumble noise parameterized with the Euler-Mascheroni constant to the pairwise edge weights and then estimating the potential of the MAP clustering, this method is able to approximate the log partition function. To find an approximate MAP, we build a tree by recursively applying sparsest cut and then select the tree consistent partition that has the highest potential. We refer to the method below as *P&M*.

Hierarchical Clustering Inference Using a tree data structure, as described in Section 2.2.2, this method computes a distribution over clusterings. Recall that in

[3] `https://archive.ics.uci.edu/ml/datasets/zoo`

	$\Delta \log Z$		
	0-Tree	Slab Tree	$\log Z$
Zoo-10	8.516	0.896	10.458
Bio-11	9.348	4.508	13.429
Coref-10	10.650	0.673	12.446

Figure 4.1: Absolute difference from the exact log partition function for the Zero-Tree (0-Tree) and Slab Tree approaches. The last column lists the exact log partition function value.

this approach, all partitions that are not tree-consistent are given 0 probability mass. We build the cluster tree by recursively applying a sparsest cut solver. We refer to this method as *0-Tree*.

Slab Tree The approach we propose in this work, using a tree structure built with recursive sparsest cut. The partition function at each node in the slab tree is computed using Perturb and MAP (*P&M*), as described above.

Note that we chose to compare slab trees with Perturb-and-MAP, as the later has been successfully applied to approximating the partition function[38]. We do not compare with additional approximate normalizer methods since we are able to analytically describe the relationship (with respect to total variation divergence from the exact distribution) between the slab tree approach and using an approximate normalizer (see Proposition 10).

4.2.3 Results

In Figure 4.1 we see the accuracy of the predicted log-partition functions on small datasets. The absolute difference between the 0-Tree and the exact partition function observed in the Zoo-10 data is the same order of magnitude as the partition function itself. Using the slab tree approach, with the partition function approximated using the Perturb and MAP approach, the difference between the difference from the exact partition function is much smaller than using the 0-tree. While the difference observed

	P&M	0-Tree	Slab Tree
Zoo-10	0.896	9.899	0.995
Bio-11	4.508	9.349	0.270
Coref-10	0.981	10.990	1.814

Figure 4.2: Mean absolute error between the exact and approximate log-probabilities computed over all clusterings, for the Perturb and MAP (P&M), Zero tree (0-Tree), and Slab Tree approaches.

in Zoo-10 and Coref-10 is on the order of 5-8 %, the difference for Bio-11 is much greater, on the order of 30 %.

In Figure 4.2 we evaluate the mean absolute error of each methods' approximate log probability of a clustering, i.e., $\frac{\sum_{\mathcal{C}\in\mathbb{C}} |\log \hat{P}(\mathcal{C}) - \log P(\mathcal{C})|}{|\mathbb{C}|}$, where $\log \hat{P}(C)$ is the predicted log probability and $\log P(C)$ is the true log probability. Note that this quantity is similar, but not the same as the $\|L\|_1$ distance, which, for the P&M approach, can be derived from the absolute differences in Table 4.1 as described in Section 4.1.2.4. The slab tree method results in a major improvement in divergence compared with the 0-tree approach. In the cases where the partition function estimates are close to the exact value, Zoo-10 and Coref-10, Perturb and MAP performs somewhat better than the slab tree approach. In the case where the partition function estimate is relatively poor, Bio-11, the slab tree approach provides a substantial improvement over Perturb and MAP.

4.3 Related Work

We are not aware of any previous work transforming approximations of partition functions of distributions of clusterings into compact representations of true probability distributions, as is done with slab trees. There has been some work on compactly representing distributions over clusterings, using either trees or DAGs, as well as estimating the partition function in this context. There has also been a substantial

work in estimating partition functions in general graphical models[4]. The most directly related work to slab trees has been on computing distributions over clusterings. As mentioned in [31], trees are special cases of sparse trellises, which are directed acyclic graphs used for a dynamic programming approach to computing the partition function exactly [31]. In [36] the authors propose dynamic programming to compute the partition function for clusterings of size k, however there is redundancy in the computation. [71] and [35] improve on this but take the same general approach. [42] compute the exact partition function using convolutions on a Mobius strip. These approaches find the exact partition function, rather than use an estimate of the function to represent the distribution. Crucially, the complexity of these approaches are exponential in the size of the dataset, making them appropriate only for small datasets.

Distributions over clusterings or partitions of a set of points are frequently modeled with Bayesian non-parameteric models such as Dirichlet Process Mixture Models [3] and equivalent statistical models such as Chinese Restaurant processes [2] and the Stick-Breaking Process [58]. In these approaches, the posterior distribution is a distribution over clusterings. These methods differ from our work in the likelihood and prior used. Our work uses a correlation clustering objective, while these works most often use parametric densities such as Gaussians or multinomials. We use a uniform prior over clusterings, while these methods typically a rich-get-richer objective as in the aforementioned methods. Furthermore, the posterior distribution typically a distribution over the parametric densities. Similar to our work Bayesian Hierarchical Clustering approximates the posterior distribution of a Dirichlet Process Mixture

[4]It may be worth noting that there is not a clear equivalent graphical model for the general energy-based clustering models we address; in the cases where there is an equivalent graphical model, e.g., in correlation clustering, the graph is fully connected (thus results relying on sparseness are of limited use here).

Model using a tree structure [33]. Only tree consistent partitions are given non-zero probability in this approach.

More generally there are several approximation methods for approximating the partition function for graphical models. [57] use a perturbation based approaches. MCMC has been a popular approach for this purpose [59] , sometimes utilizing various improvements, such as Langevin dynamics [48] to improve performance. [52] uses Gibbs sampling methods and there have been variational methods [74] proposed to provide bounds on the partition function as well.

CHAPTER 5

EXACT DISTRIBUTIONS OVER HIERARCHICAL CLUSTERINGS

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 [11, 23] for more details and proof). To overcome the computational burden, we introduce a cluster trellis data structure [32] for hierarchical clustering. We describe how this data structure enables us to use dynamic programming algorithms to exactly compute MAP structures and the partition function. Our algorithms compute these quantities without having to iterate over each possible hierarchy in the $\mathcal{O}(3^N)$ time. While still exponential, this is orders of magnitude faster than enumerating all trees and is to our knowledge the fastest exact MAP / partition function result (See §5.1.2 and §5.1.1 for proofs). Furthermore, we demonstrate how our methods can be used to sample structures from $P(\mathtt{H}|X)$, compute marginal probabilities of subtrees, all without enumerating the complete set of hierarchical clusterings.

5.1 Algorithms and Analysis

5.1.1 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 1.2. The trellis is used to facilitate 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,

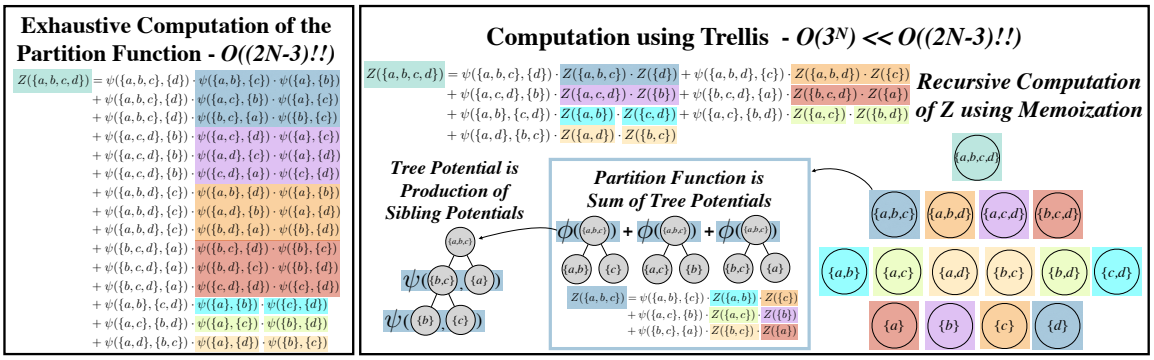


Figure 5.1: **Computing the partition function**. An example of using a trellis to compute the distribution over hierarchical clusterings of the dataset $\{a, b, c, d\}$. The left panel shows the exhaustive computation of the partition function, consisting of the summation of $(2 \cdot 4 - 3)!!$ potential equations, one for each of the $5!! = 15$ trees rooted at $\{a, b, c, d\}$. The right panel shows the computation of the partition function using the corresponding 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 that are computed with and stored in the trellis.

Proposition 11. *For any $x \in X$, the hierarchical partition function can be written recursively, as $Z(X) = \sum_{\mathtt{H} \in \mathcal{H}(X)} \mathcal{E}(\mathtt{H}) = \sum_{X_i \in X_x} \psi(X_i, X \setminus X_i) \cdot Z(X_i) \cdot Z(X \setminus X_i)$ where X_x is the set of all clusters containing the element x (omitting X), i.e,. $X_x = \{X_j : X_j \in 2^X \setminus X \wedge x \in X_j\}$.*

Proof. Given a dataset X, pick an element $x \in X$. We consider all possible Ω clusters $X_L^\omega \subset X$ that contain 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 potentials of all possible hierarchical clusterings $\mathcal{H}_X = \{\mathtt{H}^m\}_{m=1}^M$,

$$\begin{aligned} Z(X) &= \sum_{m=1}^{M} \mathcal{E}(\mathtt{H}^m(X)) \\ &= \sum_{m=1}^{M} \psi(X_L^m, X_R^m)\ \mathcal{E}(\mathtt{H}^m(X_L^m))\ \mathcal{E}(\mathtt{H}^m(X_R^m)) \end{aligned} \tag{5.1}$$

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. 5.1 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$ [1],

$$\begin{aligned} Z(X) &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega) \sum_{j=1}^{J} \mathcal{E}(\mathtt{H}^j(X_L^\omega)) \sum_{k=1}^{K} \mathcal{E}(\mathtt{H}^k(X_R^\omega)) \\ &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z(X_L^\omega)\ Z(X_R^\omega) \end{aligned} \tag{5.2}$$

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

□

Algorithm 5 describes how to efficiently compute the partition function using the trellis by making use of Proposition 11. Our algorithm works by first setting the partition function of the leaf nodes in the trellis to 1. Algorithm 5 is described in a recursive way. It starts by picking any element in the dataset, x_i, and the complement containing all points other than x_i, $X \setminus x_i$. It then considers all clusters, X_j, of the powerset of $X \setminus x_i$, i.e., $X_j \in 2^{X_j \setminus x_i}$). For the cluster $X_i = X_j \cup \{x_i\}$, the partition function is computed (memoized, recursively) for X_i and its complement, thus enabling the application of Proposition 11 to get $Z(X)$. The algorithm can equivalently 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

[1] 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.

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

Algorithm 5 `PartitionFunction`(X)

Pick $x_i \in X$ and set $Z(X) \leftarrow 0$
for X_j in $2^{X \setminus \{x_i\}}$ **do**
  $X_i \leftarrow X_j \bigcup \{x_i\}$
  if $Z(\mathbb{V}(X_i))$ not set **then** $Z(\mathbb{V}(X_i)) \leftarrow$ `PartitionFunction`(X_i)
  if $Z(\mathbb{V}(X \setminus X_i))$ not set **then** $Z(\mathbb{V}(X \setminus X_i) \leftarrow$ `PartitionFunction`$(X \setminus X_i)$
  $Z(X) \leftarrow Z(X) + \psi(X_i, X \setminus X_i) \cdot Z(\mathbb{V}(X_i)) \cdot Z(\mathbb{V}(X \setminus X_i))$
return $Z(X)$

the partition functions in this order, whenever computing the partition function of a given node in the trellis, the partition functions of all of the descendent nodes will have already been computed and memoized. In Figure 5.1, 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. We have the following complexity result for the algorithm:

Theorem 3. *For a given dataset X of N elements, Algorithm 5 computes $Z(X)$ in $\mathcal{O}(3^N)$ time.*

Proof. 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 $\sum_{i=1}^{N} 2^i \binom{n}{i} = 3^N - 1$. □

The time-complexity of the algorithm is $\mathcal{O}(3^N)$, since the partition function for each node in the trellis is computed and the descendent nodes' partition functions are always pre-computed due to the order of computation. This is significantly smaller than the $(2N-3)!!$ possible hierarchies.

5.1.2 Computing the MAP Hierarchical Clustering

Similar to other dynamic programming algorithms, such as Viterbi, we can adapt Algorithm 5 in order to find the MAP hierarchical clustering. The MAP clustering for dataset X, $\mathtt{H}^\star(X)$, is $\mathtt{H}^\star(X) = \text{argmax}_{\mathtt{H}\in\mathcal{H}(X)} P(\mathtt{H}|X) = \text{argmax}_{\mathtt{H}\in\mathcal{H}(X)} \phi(\mathtt{H})$. As in the partition function, we can use a recursive memoized technique. Each node will store a value for the MAP, denoted $\phi(\mathtt{H}^\star(X))$ and a backpointer $\Xi(\mathtt{H}^\star(X))$. To use the recursive technique we use the following Proposition for correctness of the recursion.

Proposition 12. *For any $x \in X$, let $X_x = \{X_j : X_j \in 2^X \setminus X \wedge x \in X_j\}$, then $\phi(\mathtt{H}^\star(X)) = \max_{X_i\in X_x} \psi(X_i, X \setminus X_i) \cdot \phi(\mathtt{H}^\star(X_i)) \cdot \phi(\mathtt{H}^\star(X\setminus X_i))$.*

Proof. We proceed in a similar way as detailed in Proposition 11 , as follows. Given a dataset X, pick an element $x \in X$. We consider all possible Ω clusters $X_L^\omega \subset X$ that contain 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 $\mathcal{E}(\mathtt{H}^*(X))$ can be computed recursively in terms of $\mathcal{E}(\mathtt{H}^*(X_L^\omega))$ and $\mathcal{E}(\mathtt{H}^*(X_R^\omega))$.

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

$$\begin{aligned}\mathcal{E}(\mathtt{H}^*(X)) &= \max_{m\in M} \mathcal{E}(\mathtt{H}^m(X)) \\ &= \max_{m\in M} \psi(X_L^m, X_R^m)\ \mathcal{E}(\mathtt{H}^m(X_L^m))\ \mathcal{E}(\mathtt{H}^m(X_R^m)) \qquad (5.3)\end{aligned}$$

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. 5.3 grouping together all the hierarchies $\mathtt{H}^i$ that have the same clusters $\{X_L^m, X_R^m\}$, as follows

$$\begin{aligned}\mathcal{E}(\mathtt{H}^*(X)) = &\max_{\omega\in\Omega}\Big(\psi(X_L^\omega, X_R^\omega) * \\ &\max_{j\in J}\mathcal{E}(\mathtt{H}^j(X_L^\omega))\ \max_{k\in K}\mathcal{E}(\mathtt{H}^k(X_R^\omega))\Big) \\ = &\max_{\omega\in\Omega}\psi(X_L^\omega, X_R^\omega)\ \mathcal{E}(\mathtt{H}^*(X_L^\omega))\ \mathcal{E}(\mathtt{H}^*(X_R^\omega)) \qquad (5.4)\end{aligned}$$

with $M = \Omega \cdot J \cdot K$. Thus, $\mathcal{E}(\mathtt{H}^*(X))$ of a cluster X can be written recursively in terms of the MAP values of the sub-clusters of X [3].

□

As in the partition function algorithm described in Section 5.1.1, the time complexity for finding the MAP clustering is $\mathcal{O}(3^N)$. To compute the maximal likelihood hierarchical clustering, the maximal potential of the sub-hierarchy rooted at each node is computed, rather than 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, follows:

Proof. 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 $\sum_{i=1}^{N} 2^i\binom{n}{i} = 3^N - 1$.

□

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

Algorithm 6 `MAP`(X)

if $\phi(\mathbb{V}(X))$ set **then**
  return $\phi(X), \Xi(X)$
Pick $x_i \in X$
$\phi(X) \leftarrow -\infty$
$\Xi(X) \leftarrow$ `null` {Backpointer to give MAP tree structure.}
for X_j in $2^{X \setminus \{x_i\}}$ **do**
  $X_i \leftarrow X_j \bigcup \{x_i\}$
  $t \leftarrow \psi(X_i, X \setminus X_i) \cdot \phi(\mathbb{V}(X_i)) \cdot \phi(\mathbb{V}(X \setminus X_i))$
  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 $\phi(X), \Xi(X)$

5.1.3 Computing the Number of Hierarchical Clusterings

We note for interest that we're able to count the total number of hierarchies using the trellis[4]. 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

$$N_p^i = N_{X_L}^i \cdot N_{X_R}^i \tag{5.5}$$

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.

5.1.4 Computing Marginal Probabilities

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., $\mathtt{H}_i \in \mathcal{H}(X_i)$, 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 $\mathtt{H}_i \subset \mathtt{H}$

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

indicates that $\mathtt{H}_i$ is a subtree of $\mathtt{H}$. The second is for a given cluster, X_i, defined as $P(X_i|X) = \sum_{\mathtt{H}_i \in \mathcal{H}(X_i)} P(\mathtt{H}_i|X)$.

Proposition 13. *The value of $P(\mathtt{H}_i|X_i)$ can be computed using the same algorithm used for the partition function, except by first merging $X(\mathtt{H}_i|X)$ into a single leaf node, but using $\phi(X(\mathtt{H}_i))$ for the potential of the newly merged leaf.*

Proof. 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:

$$P(\mathtt{H}_i|X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} P(\mathtt{H}|X) = \sum_{\mathtt{H} \in A(\mathtt{H}_i)} \frac{\phi(\mathtt{H}(X))}{Z} = \frac{Z_{\mathtt{H}_i}(X)}{Z} \tag{5.6}$$

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

$$\begin{aligned} Z_{\mathtt{H}_i}(X) &= \sum_{m=1}^{|A(\mathtt{H}_i)|} \mathcal{E}(\mathtt{H}^m(X)) \\ &= \sum_{m=1}^{|A(\mathtt{H}_i)|} \psi(X_L^m, X_R^m)\ \mathcal{E}(\mathtt{H}^m(X_L^m))\ \mathcal{E}(\mathtt{H}^m(X_R^m)) \end{aligned} \tag{5.7}$$

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. 5.7 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 $X_i \subseteq X_L$.

$$
\begin{aligned}
Z_{\mathtt{H}_i}(X) &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega) \sum_{j=1}^{J} \mathcal{E}(\mathtt{H}^j(X_L^\omega)) \sum_{k=1}^{K} \mathcal{E}(\mathtt{H}^k(X_R^\omega)) \\
&= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z_{\mathtt{H}_i}(X_L^\omega)\ Z(X_R^\omega)
\end{aligned}
\tag{5.8}
$$

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 $Z_{\mathtt{H}_i}(X_i) = \mathcal{E}(\mathtt{H}(X_i))$. □

Proposition 14. *The value of $P(X_i|X)$, can be computed using the same algorithm used for the partition function, except by first merging $X(\mathtt{H}_i|X)$ into a single leaf node, but using $Z(X_i)$ for the potential of the newly merged leaf.*

Proof. 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:

$$
P(X_i|X) = \sum_{\mathtt{H}\in A(X_i)} P(\mathtt{H}|X) = \sum_{\mathtt{H}\in A(X_i)} \frac{\phi(\mathtt{H}(X))}{Z} = \frac{Z_{X_i}(X)}{Z} \tag{5.9}
$$

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

$$
\begin{aligned}
Z_{X_i}(X) &= \sum_{m=1}^{|A(X_i)|} \mathcal{E}(\mathtt{H}^m(X)) \\
&= \sum_{m=1}^{|A(X_i)|} \psi(X_L^m, X_R^m)\ \mathcal{E}(\mathtt{H}^m(X_L^m))\ \mathcal{E}(\mathtt{H}^m(X_R^m))
\end{aligned}
\tag{5.10}
$$

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.

5.10 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 $X_i \subseteq X_L$.

$$\begin{aligned} Z_{X_i}(X) &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega) \sum_{j=1}^{J} \mathcal{E}(\mathtt{H}^j(X_L^\omega)) \sum_{k=1}^{K} \mathcal{E}(\mathtt{H}^k(X_R^\omega)) \\ &= \sum_{\omega=1}^{\Omega} \psi(X_L^\omega, X_R^\omega)\ Z_{X_i}(X_L^\omega)\ Z(X_R^\omega) \end{aligned} \tag{5.11}$$

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 $Z_{X_i}(X_i) = Z(X_i)$. □

5.1.5 Sampling from the Distribution of Hierarchical Clusterings

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

The sampling procedure will build a tree structure in a top-down way. We start with the cluster of all the elements, X, and then sample one child of that cluster, $X_L \subset X$, (Eq. 5.12) and set the other to be the complement of the child with respect to the parent, i.e., $X \setminus X_L$. This procedure repeats 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:

$$p(X_L|X_p) = \frac{1}{Z(X_p)} \cdot \psi(X_L, X_p \backslash X_L)\ \cdot Z(X_L) \cdot Z(X_p \backslash X_L). \tag{5.12}$$

Pseudocode for this algorithm is given in Algorithm 7.

Theorem 4. $\mathtt{Sample}(X)$ *(Alg. 7) gives samples from* $P(\mathtt{H}|X)$.

Proof. We want to show that drawing samples of trees using Algorithm 7 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 7.

Recall from Definition 4 we have:

$$P(\mathtt{H}|X) = \frac{1}{Z(X)} \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} \psi(X_L, X_R) \tag{5.13}$$

We can equivalently write this as:

$$P(\mathtt{H}|X) = \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} \frac{1}{Z(X_L \cup X_R)} \cdot \psi(X_L, X_R) \cdot Z(X_L) \cdot Z(X_R) \tag{5.14}$$

To understand why this can be written this way, observe that for for each in $\mathsf{sibs}(H)$ (i.e., internal nodes such as X_L and X_R 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 nodes X_L, X_R and their children X_{LL}, X_{LR} and X_{RL} and X_{RR} respectively:

$$\frac{1}{Z(X_p)} \psi(X_L, X_R)\ Z(X_L)\ Z(X_R) \cdot \frac{1}{Z(X_L)} \psi(X_{LL}, X_{LR})\ Z(X_{LL})\ Z(X_{LR}) \cdot \frac{1}{Z(X_R)} \psi(X_{RL}, X_{RR})\ Z(X_{RL})\ Z(X_{RR}) \tag{5.15}$$

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

Algorithm 7 $\texttt{Sample}(X)$

if $|X| = 1$ **return** $\{X\}$
Sample X_L from $p(X_i|X)$ (Eq. 5.12).
return $\{X_L,\ X \setminus X_L\} \cup \texttt{Sample}(X_L) \cup \texttt{Sample}(X \setminus X_L)$

Next, we observe that Eq. 5.14 can be re-written in terms of Equation 5.12 which defines $p(X_L|X_L \cup X_R)$:

$$P(\mathtt{H}|X) = \prod_{X_L, X_R \in \mathsf{sibs}(\mathtt{H})} p(X_L|X_L \cup X_R) \tag{5.16}$$

Algorithm 7 applies Eq. 5.12 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. 5.16.

□

This algorithm is notable in that it does not require computing a categorical distribution over all trees and samples exactly according to $P(\mathtt{H}|X)$.

5.1.6 Trellis Construction

For completeness, we note that Algorithm 3 for building trellises presented in 3.1.5 can be used to build trellises for hierarchical clustering as well.

5.2 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 [24]. 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 use a simulation model for cascades of particle physics decays in jet physics that provides

ground truth hierarchies facilitating evaluation. We additionally demonstrate the use of the sampling procedure (§5.1.5) in this domain. In cancer genomics, we show how we can model subtypes of cancer, which can help determine prognosis and treatment plans. Lastly, 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.

5.2.1 Jet Physics and Hierarchical Clusterings

5.2.1.0.1 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 generated by the showering process. While the latent showering process is unobserved, it is described by quantum chromodynamics (QCD).

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 [46].

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* [10], 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 5.2.

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 [21]. 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 [21].

5.2.1.0.2 Data and Methods In this paper, we proposed a new method to efficiently find the MAP hierarchical clustering, partition function Z, and compute an estimate for the posterior distribution over all possible hierarchical clusterings from sampling. We will compare the trellis 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 [20] 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

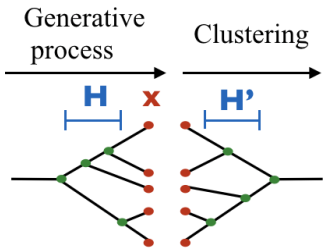


Figure 5.2: Schematic representation of the tree structure of a sample jet generated with Ginkgo and the clustered tree for some clustering algorithm. For a given algorithm, z labels the different variables that determine the latent structure of the tree. The tree leaves x are labeled in red and the inner nodes in green.

state particles (inner nodes) in Ginkgo are represented by a four dimensional energy-momentum vector.

Next, we review new algorithms to cluster jets based on the joint likelihood of the jet binary tree in Ginkgo, introduced in [22]. In this work the authors explore algorithms that aim to obtain the maximum likelihood estimate (MLE) or MAP for the latent structure of a jet. In this approach, the tree latent structure z_{shower} is fixed by the algorithm. In particular, greedy and beam search algorithms are studied. 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 [9] for details about the different orderings of the internal nodes of the tree). This approach significantly improved the performance of the beam search algorithm in terms of finding the MLE.

5.2.1.0.3 Results In this section we show results for the implementation of the trellis algorithm on a jet physics dataset of 5000 Ginkgo [21] jets with a number of leaves between 5 and 10, and we refer to it as Ginkgo510. We start by comparing in Table 6.1 the mean difference among the MAP values for the hierarchies likelihood obtained with the trellis, beam search and greedy algorithms. We see that the likelihood of the trees increase from greedy to beam search to the trellis one, as expected. We use beam search as a baseline to estimate the MAP value, which typically has a good performance for trees with up to about 10 leaves, but as we see in Table 6.1, the trellis MAP value is greater. Next, in Figure 5.3 we show a plot of the partition function versus the MAP 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. We want to

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

Table 5.1: 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.

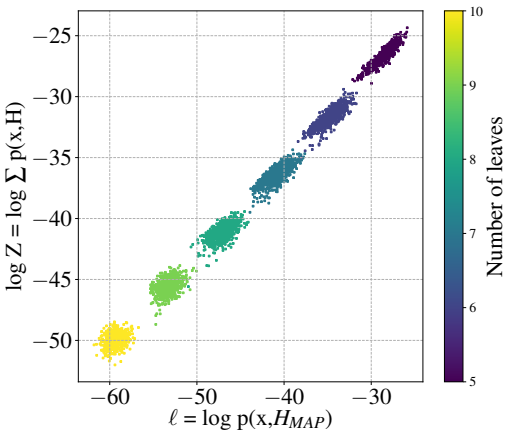


Figure 5.3: Scatter plot of the partition function Z vs. the trellis MAP ℓ for the Ginkgo510 dataset, with up to 10 leaves (jet constituents). The color indicates the number of leaves of each hierarchical clustering. There appears to be a correlation between Z and the MAP.

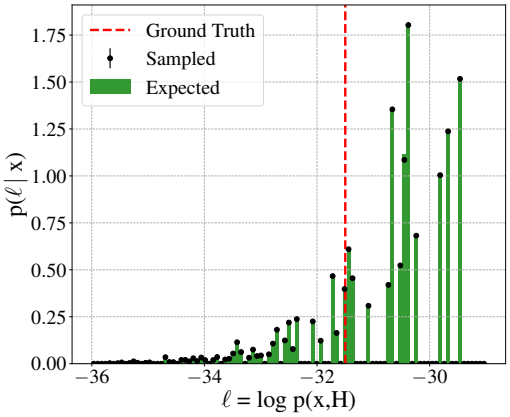


Figure 5.4: Comparison of the posterior distribution for a specific jet with five leaves for sampling 10^5 hierarchies using Alg 7 (black dots with small error bars) and expected posterior distribution (in green). The plots show the discrete nature of the distribution. The log likelihood for the ground truth tree is a vertical dashed red line.

emphasize that the implementation of the trellis algorithm allows us to access the partition function.

Finally, we show the implementation of the sampling procedure introduced in section 5.1.5. We compare the sampled posterior distribution with respect to the expected one (as explained below), conditioned on a set of five leaves. We show in Figure 6.4 the results from sampling 10^5 hierarchies (black dots) and the expected distribution (green) for the likelihood of each hierarchy. 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 6.4. There is an excellent agreement between the sampled and the expected distributions.

5.2.2 Cancer Genomics and Hierarchical Clusterings

5.2.2.0.1 Background Hierarchical clustering is a common clustering approach for gene expression data [65]. 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.

5.2.2.0.2 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 [70]. 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. The Pearson correlation coefficient was used for the clustering metric for the PAM50 data set experiments. 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. However, the correlations among subsampled `pam50` ($n = 12$) data set are all positive. To allow more

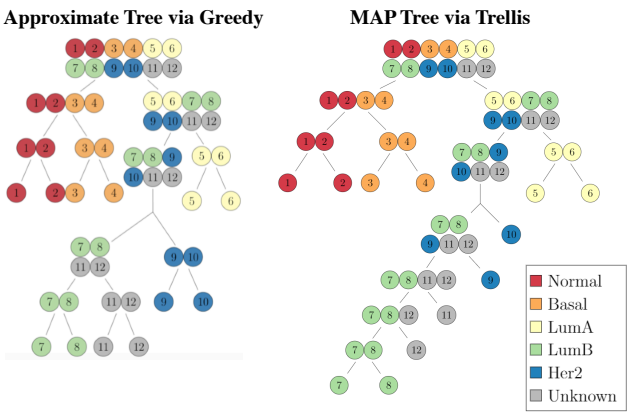


Figure 5.5: **Cancer Genomics**. Comparison of trees from greedy hierarchical clustering (left) and exact MAP clustering using the trellis (right) on the subsampled `pam50` data set. The colors indicate subtypes of breast cancer (grey if unknown). Though both appear to assign unknown samples to LumB, the right tree positions the unknown samples closer to the Her2 samples.

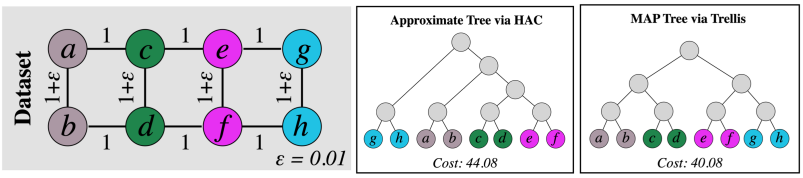


Figure 5.6: **Dasgupta's Cost**. Comparison between MAP tree found using the trellis and the tree found by agglomerative clustering for a graph that is known to be difficult for with greedy methods.

balanced weights among vertices, we transform the weights using $w'_{uv} = \tan\left(w_{uv}\frac{\pi}{2}\right)$ and then subtracted the average of all w'_{uv}.

5.2.2.0.3 Results Figure 5.5 displays the greedy hierarchical clustering tree and the MAP tree with transformed weights for the twelve samples selected from the pam50 dataset. 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.

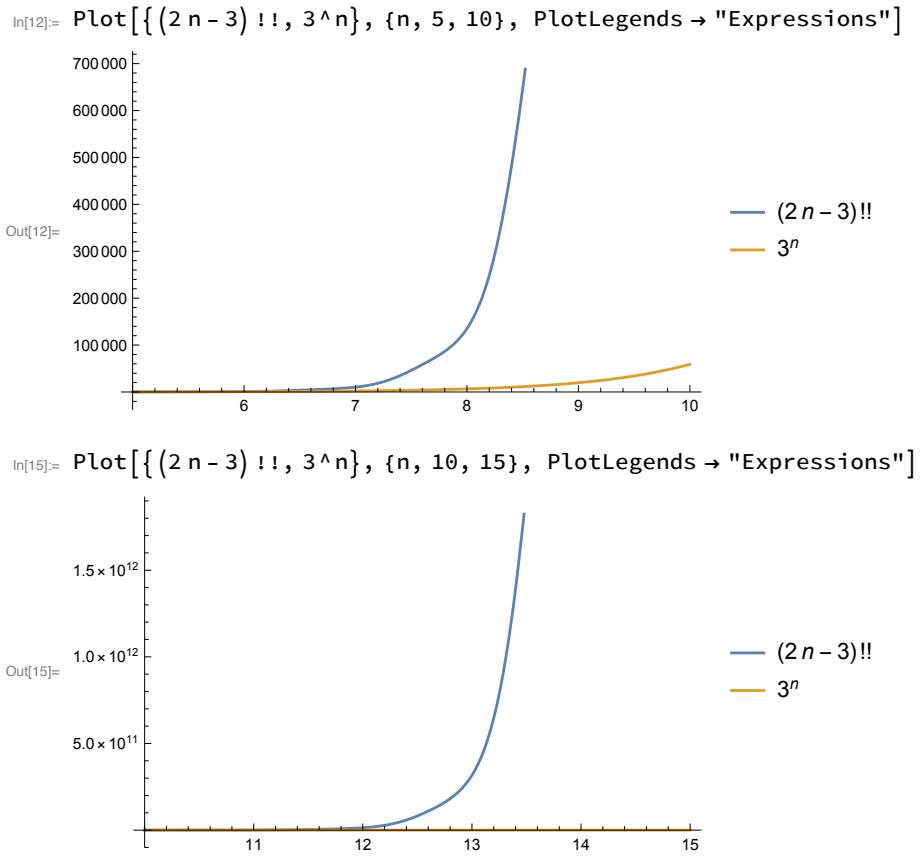


Figure 5.7: Comparison of the the number of trees vs complexity of trellis algorithms.

5.2.3 Dasgupta's Cost

Figure 5.6 gives an example graph, as proposed by [14] to bound average-linkage performance, following a model for which greedy methods are known to be approximate with respect to Dasgupta's cost [53, 18]. We run greedy agglomerative clustering and trellis-based MAP procedure (Eq. 1.7). 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.

5.2.4 Runtime Asymptotics Plots

See Figure 5.7 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.

5.3 Related Work

Often, probabilistic approaches, such as diffusion trees [54, 40] and coalescent models [69, 9, 34], model which tree structures are likely for a given dataset. For instance, in particle physics generative models of trees are used to model jets [10], and similarly coalescent models have been used in phylogenetics [67]. Inference in these approaches is done by approximate, rather than exact, methods such as greedy best-first, beam-search, sequential Monte Carlo [75], and MCMC [54] and lead to local optima. These methods do not have efficient ways to compute an exact normalized distribution over all tree structures.

Despite this, modeling distributions over tree structures has been the subject of a large body of work, including various types of Bayesian non-parametric models, e.g., [40, 34, 7, 56, 30]. These methods only support using parametric distributions to define emission probabilities and do not support the general family of potential functions as in our method. 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. Bootstrapping methods, such as [68], represent uncertainty in hierarchical clustering, however they approximate statistics of interest through repeatedly (re-)sampling from the empirical distribution. Exact methods like the one we propose have not, to our knowledge, been proposed. See Chapter 6 in general, and Section 6.3 in particular, for a more in depth discussion of approximate distributions over hierarchical clusterings.

Recent work [38, 42, 31] has studied distributions over flat clusterings and showed that dynamic programming can be used to both efficiently compute the partition function as well as find the MAP clustering [71, 32]. See Chapter 3 in general, and Section 3.4 in particular, for a more in depth discussion of exact distributions over flat clusterings

CHAPTER 6

APPROXIMATING DISTRIBUTIONS OVER HIERARCHICAL CLUSTERINGS

Hierarchical clustering is a classic unsupervised learning task with numerous applications. In settings where an potential function or likelihood can be assigned to candidate clusters, the clustering problem can be cast in probabilistic terms. Approximate inference methods often use greedy/beam search to explore candidate hierarchies. Exact methods such as the hierarchical trellis (Greenberg et al, 2020) explore the entire space of trees using a dynamic programming approach. While this approach is efficient compared to the double factorial growth in the number of possible hierarchical clusterings[1], the scaling of this algorithm is limited to small datasets.

Thus in this section, we utilize a sparse trellis, which allows to scale to much larger datasets by controlling the sparsity index, i.e. the fraction of hierarchies we consider from the total of $(2N-3)!!$. Most hierarchies have likelihood values orders of magnitude smaller than the MAP clustering making their contribution to the partition function negligible [2]. As a result, if we build a sparse trellis that considers the most relevant hierarchies, we could find approximate solutions for the MAP values and partition function for datasets where implementing the exact trellis is not feasible. The algorithms are similar to the exact cluster trellis ones, though there are differences in the implementation.

---

[1]The number of hierarchical clusterings over N elements grows as $(2N-3)!!$ (see [11, 23])

[2]In general, low likelihood is equivalent to large potentials. However, we refer to likelihood as this is directly connected to the jet physics example used as a case study in this chapter.

Additionally, we show how the sparse trellis can be used to sample structures from $P(\mathtt{H}|X)$ and compute the posterior distribution for the likelihood of the hierarchies conditioned on a set of leaves, without enumerating the full set of hierarchical clusterings.

6.1 Algorithms and Analysis

With an amendment to the edges of the trellis data structure and a corresponding reformulation of the algorithms for hierarchical clusterings presented in Chapter 3, we are able to approximate the partition function, MAP clustering, and the marginal probability (of a given sub-hierarchy or a cluster). Our approach is similar to one we utilized for flat clustering–making use of sparse trellises. Unlike sparse trellises for flat clustering, sparse trellises for hierarchical clustering need not be closed under recursive complement, instead having the lesser requirement that only valid hierarchies are represented by the edge structure. This crucial reliance on the trellis edges as part of these algorithms is the primary difference between them and those presented in earlier chapters, and is what makes them practical without the need for closure under recursive complement.

6.1.1 Approximating the Partition Function

In order to approximate the partition function, we make use of the sparse trellises for hierarchical clustering described in Section 2.2. Our algorithm is analogous to Algorithm 5, given in Section 5.1.1, with the essential difference that when computing the partition function for vertex $\mathbb{V}$, the algorithm iterates over the edge structure.

A sparse trellis for hierarchical clustering, $\widehat{\mathbb{T}}$, represents a set of hierarchical clusterings through its edge structure. Recall, each vertex in vertex, $\mathbb{V}$, in sparse trellis $\widehat{\mathbb{T}}$ has edges to a subset of all two-partitions of $X(\mathbb{V})$, and we denote this subset $\mathsf{ch}(\mathbb{V})$. Any hierarchical clustering, $\mathtt{H} \in \mathcal{H}$, that contains an edge from $X(\mathbb{V})$ to $X_i \in \mathbb{P}(X(\mathbb{V}))$

s.t. $X_i \notin \mathsf{ch}(\mathbb{V})$ is said to be *inconsistent* with sparse trellis $\widehat{\mathbb{T}}$, otherwise $\mathtt{H}$ is *consistent* with $\widehat{\mathbb{T}}$ (or sparse-trellis-consistent). We denote the set of all hierarchical clusterings consistent with $\widehat{\mathbb{T}}$, as $\mathcal{H}(\widehat{\mathbb{T}})$.

Proposition 15. *The partition function over the set of sparse-trellis-consistent hierarchical clusterings, $\mathcal{H}(\widehat{\mathbb{T}})$, is given by:*

$$Z(X) = \sum_{\mathtt{H} \in \mathcal{H}(\widehat{\mathbb{T}})} \phi(\mathtt{H}) = \sum_{X_L, X_R \in \mathsf{ch}(\mathbb{V}(X))} \psi(X_L, X_R) \cdot Z(X_L) \cdot Z(X_R) \tag{6.1}$$

Proof. The proof is the same as given for Proposition 11, only instead of summing over all two-partitions, the sum is over all two-partitions that are present in the sparse trellis. Any two-partition of $\mathbb{V}$ missing from $\mathsf{ch}(\mathbb{V}(X))$ can not be part of a hierarchical clustering that is consistent with $\widehat{\mathbb{T}}$, thus such trees get 0 potential. □

Analogous to Section 5.1.1, we describe an algorithm to efficiently approximate the partition function given a sparse trellis by making use of Proposition 15. Similar to Algorithm 5, it works by setting the partition function of the leaf nodes in the trellis to 1, then following the edges from each leaf to their set of parents, and then computing the partition function for those vertices using Proposition 15. The partition functions for every vertex is computed and memoized in this way, from the leafs to the root. Note that when computing the partition function for any vertex, $\mathbb{V}$ in $\widehat{\mathbb{T}}$, the partition functions for all of $\mathbb{V}$'s children has already been memoized. See Algorithm 8 for pseudocode.

As in flat-clustering sparse trellises, our algorithm is a function of the size of the trellis, though in this case, specifically the number of edges. The time complexity for our algorithm is:

Theorem 5. *For a given dataset X of N elements, Algorithm 8 computes $Z(X)$ in $\mathcal{O}(|E(\widehat{\mathbb{T}})|)$ time.*

Algorithm 8 $\texttt{PartitionFunction}(X)$

Set $Z(X) \leftarrow 0$
for X_L, X_R in $\mathsf{ch}(\mathbb{V}(X))$ **do**
  if $Z(\mathbb{V}(X_L))$ not set **then** $Z(\mathbb{V}(X_L)) \leftarrow \texttt{PartitionFunction}(X_L)$
  if $Z(\mathbb{V}(X_R))$ not set **then** $Z(\mathbb{V}(X_R) \leftarrow \texttt{PartitionFunction}(X_R)$
  $Z(X) \leftarrow Z(X) + \psi(X_L, X_R) \cdot Z(\mathbb{V}(X_L)) \cdot Z(\mathbb{V}(X_R))$
return $Z(X)$

Proof. The partition function is computed recursively, starting at the root, and following the edge structure down the trellis to the leaves. Each edge is followed once to a given vertex, $\mathbb{V}$ and, a constant computation (a sum and a product) at each vertex is needed. □

6.1.2 Approximating the MAP Clustering

As with Algorithm 5, we can adapt Algorithm 8 in order to approximate the MAP hierarchical clustering. The MAP clustering that is sparse-trellis-consistent with $\widehat{\mathbb{T}}$, $\mathtt{H}^\star(\widehat{\mathbb{T}})$, is $\mathtt{H}^\star(\widehat{\mathbb{T}}) = \text{argmax}_{\mathtt{H} \in \mathcal{H}(\widehat{\mathbb{T}})} \phi(\mathtt{H})$. Just as in the MAP hierarchical clustering algorithm described in Section 5.1.2, we use a recursive memoized technique, and each node stores a local MAP value, denoted $\phi(\mathtt{H}^\star_{\widehat{\mathbb{T}}}(X))$, rather than a local partition function value. We use the following Proposition for correctness of the recursion.

Proposition 16. *The MAP hierarchical clustering consistent with sparse trellis* $\widehat{\mathbb{T}}$ *is given by* $\phi(\mathtt{H}^\star_{\widehat{\mathbb{T}}}(X)) = \max_{X_i \in \mathsf{ch}(\mathbb{V}(X))} \psi(X_i, X \setminus X_i) \cdot \phi(\mathtt{H}^\star_{\widehat{\mathbb{T}}}(X_i)) \cdot \phi(\mathtt{H}^\star_{\widehat{\mathbb{T}}}(X \setminus X_i))$.

As in the partition function algorithm described in Section 6.1.1, the time complexity for finding the MAP clustering is $\mathcal{O}(|E(\widehat{\mathbb{T}})|)$.

Proof. These proofs for these are the same as for Proposition 15 and Theorem 5, replacing the partition function with the MAP clustering. □

6.1.3 Approximating Marginals

We describe how to compute the two types of marginal probabilities described in Section 5.1.4: the marginal probability of a given sub-hierarchy rooted at X_i, $P(\mathtt{H}_i | X_i)$,

and the marginal probability of a given cluster, X_i, $P(X_i|X)$. The value of $P(\mathtt{H}_i|X_i)$ can be computed using algorithm described in Section 5.1.4, except by first merging $X(\mathtt{H}_i|X)$ into a single leaf node, but using $\phi(X(\mathtt{H}_i))$ for the potential of the newly merged leaf. This assumes $\mathtt{H}_i$ is sparse-trellis-consistent with $\widehat{\mathbb{T}}$, otherwise the marginal probability is 0. The same is true for computing the value of $P(X_i|X)$, except after merging X_i into a single leaf node, the value $Z(X_i)$ should be used.

Proof. The proofs are the same as those given in Section 5.1.4, only instead of summing over all two-partitions, the sum is over all two-partitions that are present in the sparse trellis. Any two-partition missing from a $\mathsf{ch}(\mathbb{V}(\mathrm{X}))$ can not be part of a hierarchical clustering that is consistent with $\widehat{\mathbb{T}}$, thus such trees get 0 potential. □

6.1.4 Sparse Hierarchical Trellis Construction

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 the set of hierarchies consistent with the sparse trellis. The set of hierarchies can be generated however desired, e.g., using multiple different state of the art methods that return a single hierarchy, or using or more methods that can generate multiple likely hierarchies (for example, beam search), or some combination.

Note that some care must be taken when constructing a sparse trellis as the union of trees, since when a vertex is present in multiple trees: (1) it will be paired under one or more, but not necessarily all, of it ancestors, and (2) each tree has its own set of descendants of that vertex.

See Section 6.2.1 for particular building strategies used in our experiments, as well as Section 7.3 for other approaches.

6.2 Experiments

In this section, we show experiments comparing the performance of the sparse trellis to compute various probabilistic quantities that are important in real world applications in particle physics. We compare the MAP hierarchical clustering and partition function of the sparse vs the exact trellis. Also, for the MAP values, we include other approximate methods such as greedy and beam search algorithms, and analyze the improvements versus the sparsity of the trellis. Our dataset contains the ground truth hierarchies sampled from a toy model to simulate cascades of particle physics decays in jet physics. Finally, we provide an implementation of the sampling procedure to obtain the posterior distribution, detailed in section 5.1.5, and compare the sampled distribution for different ways of building the sparse trellis with the exact trellis one.

6.2.1 Sparse Trellis Building Strategies

There are 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 the map. For this paper, we ordered the leaves in increasing norm of their momentum vector $\vec{p} \in \mathbb{R}^3$ (see section 1.2 for more details about the model).

Once we choose a specific ordering, we iterate over the input trees, 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 6.1. 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.

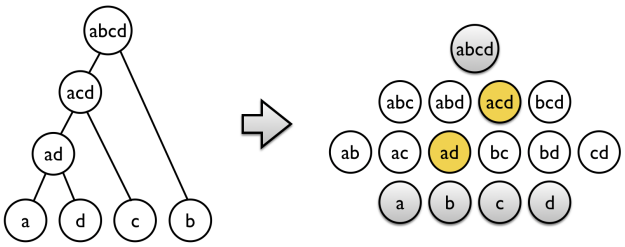


Figure 6.1: Schematic representation of how the trellis is built iterating over each tree with four leaves from a sample dataset. After every hierarchical structure is added, the final trellis is composed of the colored vertices, the leaves and the root vertex. The vertices that are not colored represent the subset of vertices of the exact trellis that are missing in the sparse case.

We present results for two distinctive procedures to build the trellis, and name the trellises according to them as Simulator and Beam Search trellis.

Simulator Trellis: Input trees are sampled from running a simulator. In this case, sample trees are expected to be around the mean of the posterior distribution. Also, we restrict the generated trees to have the same number of leaves, which is fixed for each trellis we create.

Beam Search Trellis: Input trees are obtained from running the beam search algorithm over a sample of datasets of leaves. This approach is much more general, as it could be implemented for datasets where there is no generative model. In this case, we expect the distribution for the likelihood of our input trees to be shifted toward higher values, compared to the simulator trellis. 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).

6.2.2 Binary Tree Representation of Particle Physics Jets

Detectors at the Large Hadron Collider (LHC) at CERN measure the energy (and momentum) of particles that hit them. Typically, the pattern of particle hits will have localized regions. The particles in each region are clustered and referred to as a *jet*, and can be thought of as the leaves of a binary tree. This tree is originated by a

showering process where an initial (unstable) particle (root) goes through successive binary splittings until reaching the final state particles that hit the detector and are represented by the leaves. These leaves are observed while the latent showering process, described by quantum chromodynamics (QCD), is not. As a result, there are several latent trees that correspond to a set of leaves. This representation, first suggested in [46], connects jets physics with natural language processing (NLP) and biology.

Currently, generative models in full physics simulations for the showering process that produces a set of leaves do not admit a tractable density (they are implicit models). A main problem in data analyses in collider physics deals with estimating this latent showering process. Thus, an open area of research aims to unify generation and inference, which typically requires extracting additional information from the simulator; e.g estimate the clustering history of a set of leaves (final state particles).

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 [21]. 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 [21].

6.2.3 Data & Methods

In this chapter, we proposed methods to build a sparse hierarchical trellis to efficiently find approximate values for the MAP hierarchical clustering and partition

function Z while a considering a significantly smaller number of hierarchies. We also implemented a sampling procedure over this sparse trellis that gives an estimate for the posterior distribution over all possible hierarchical clusterings.

The ground truth hierarchical clusterings of our dataset are generated with the toy generative model for jets Ginkgo [20]. We will compare the sparse trellis results for the MAP hierarchical clustering with the exact trellis, as well as approximate methods. These are implementations of greedy and beam search algorithms that aim to obtain the maximum likelihood estimate (MLE). Greedy locally maximizes the likelihood at each step. For beam search, we keep b trees (where b labels the beam size). At each step, we expand each of those trees over the top b options that maximize the clustering likelihood and keep the top b of the total of b^2 trees. Our implementation, takes b given by $\frac{N(N-1)}{2}$, with N the number of jet constituents to cluster. We avoid double counting of the different orderings of the same clustering (for specifics about the possible orderings of the internal nodes of a tree see [9]).

6.2.4 Results

In this section we compare results between the sparse trellis implemetation and the other benchmark algorithms on a jet physics dataset of 25,000 Ginkgo [21] jets with 9 leaves, and we refer to this dataset as Ginkgo9. We split our dataset into a train and a test datasets [3]. We compare in Figures 6.2 and 6.3 the results for the mean values over 100 trees of the test dataset for the sparse trellis built with two approaches, the ground truth trees (Simulator trellis) and the MLE trees obtained with beam search (BS trellis), both from the train partition of the dataset.

We start by comparing in Table 6.1 the mean difference among the MAP values for the hierarchies likelihood obtained with the Exact, BS, and Sim. trellises, as

[3]We take the terms *train* and *test* from machine learning tasks but there is no optimization method in our algorithm

Table 6.1: 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 Ginkgo9 test dataset.

	BS Trellis	Sim. Trellis	Beam Search	Greedy
Exact Trellis	0.03 ± 1.32	0.23± 1.34	0.89±0.60	1.75±0.97
BS Trellis		0.20±0.27	0.86±1.43	1.73±1.64
Sim. Trellis			0.66±1.46	1.52±1.65
Beam Search				0.86±0.97

well as beam search and greedy algorithms. We see that the potential values of the trees found using the sparse trellises are greater than beam search and greedy, with a sparsity index of only 6×10^{-2}.

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%. For this values, the trellis is very efficient and fast to run. Both sparse trellises approach the performance of the exact one, but the BS trellis does it much faster. Next, in Figure 6.3 we show a plot of the partition function versus the sparsity, on the test dataset. We see that the BS trellis approaches the exact value for the partition function faster. However, both trellises obtain a value for the partition function within 5% of the exact one with a sparsity index of 0.2 or less.

Next, we show in Figure 6.4 the posterior distribution from sampling 10^5 hierarchies with the Exact, Sim. and BS trellises. We compare the Sim. and BS trellis for sparsity values of about 0.9 and 0.1 respectively. There is an excellent agreement between the Sim. trellis and the Exact one. It is interesting to note that even though the BS trellis distribution is slightly shifted toward greater log likelihood ℓ values there is a reasonable agreement with the exact one, for a sparsity of only ~ 0.1.

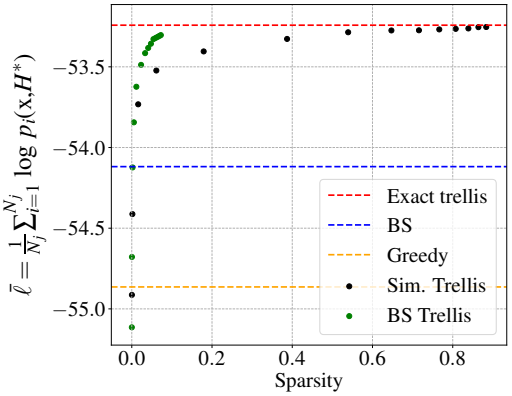


Figure 6.2: Scatter plot of the trellises MAP ℓ vs their sparsity. Each value corresponds to the mean over 100 trees of the test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises. We add the values of the exact trellis (red), beam search (blue) and greedy (orange) algorithms for comparison. The BS trellis approaches the performance of the exact one for a much smaller sparsity index.

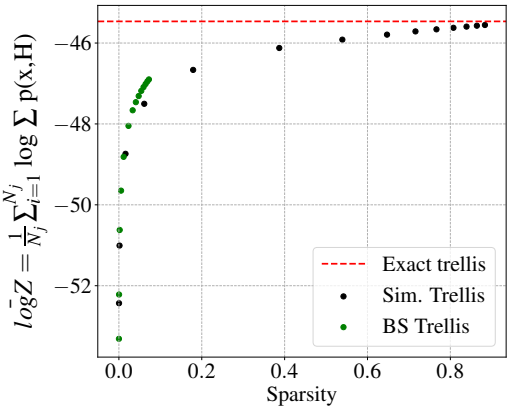


Figure 6.3: Scatter plot of the trellises partition function Z vs their sparsity. Each value corresponds to the mean over 100 trees of the test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises. We add the value of the exact trellis (red) for comparison.

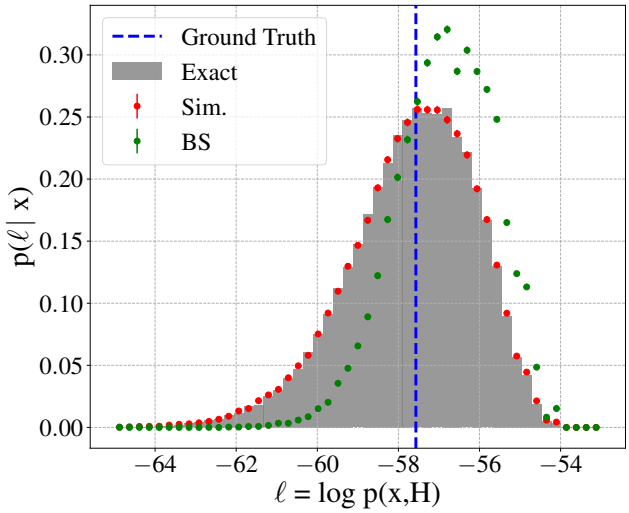


Figure 6.4: Posterior distribution of ℓ for a specific jet with five leaves. We show the distribution from sampling 10^5 hierarchies from the posterior using the procedure described in Sec. 5.1.5. We compare the distribution from the Sim. trellis (red), BS trellis (green) and Exact one (gray). The log likelihood ℓ for the ground truth hierarchical clustering is shown as a vertical dashed blue line.

6.3 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 Dirichlet and Pitman-Yor diffusion trees [54, 40], coalescent models [69, 9, 34], and in the case of grouped data, the nested Chinese restaurant processes [7] and nested hierarchical Dirichlet processes [56]. Other models, such as tree structured nested sticking breaking, provide a distribution over a different class of tree structures, one for which data can sit at internal nodes [30]. 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 probabilistic models used in our approach, which can use any scoring function to define the distribution. Factor graph-based distributions over tree structures such as [76] on the other hand support a flexible class of distributions over tree structures as in our approach.

Dasgupta [24] defines a cost function for hierarchical clustering. Much work has been done to develop approximate solution methods and related objectives [62, 19, 12, 53, 18, 14].

Bootstrapping methods, such as [68], represent uncertainty in hierarchical clustering. Unlike our approach, bootstrapping methods approximate statistics of interest through repeatedly (re-)sampling from the empirical distribution.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

As a result of the work in this thesis, we are now able to compute exact distributions and MAP values over the set of flat clusterings and the set of hierarchical clusterings for energy-based clustering models. The developed techniques can also be used approximate distributions and MAP values over these sets. There are real world applications that are able to benefit from these techniques, including particle physics at the Large Hadron Collider at CERN and cancer genomics.

We include a summary table listing the time and space complexities, as well as the approximate maximum number of elements for each of the major approaches described in this thesis. See Table 7.1.

The data structures and algorithms also lay the foundation for many new and interesting research directions and future work[1]. We share our thoughts on a few these here.

7.1 Sparse Trellises as Constraint Encoding

It is sometimes useful to consider a subset of all possible flat or hierarchical clusterings that satisfy some constraints for flat and hierarchical clustering with constraints, respectively), for example, when there is existing knowledge that would be better encoded than learned or when the application demands adjusting the probability mass of some clusterings based on some set of criteria. Encoding existing knowledge can

[1]One potential future endeavor is to unify the trellis data structures and algorithms using the approach developed for sparse hierarchical trellises.

Table 7.1: The runtime and space complexities, as well as the order of magnitude of the maximum dataset size for each of the major approaches described in this thesis. Note that the time complexity for Slab Tree Construction assumes the partition function estimate and the cut selection for any given node are provided in constant time.

	Time	Space	Max Dataset Size
Trellis Partition Function	$\mathcal{O}(3^{N-1})$	$\mathcal{O}(2^N)$	20
Trellis MAP Clustering	$\mathcal{O}(3^{N-1})$	$\mathcal{O}(2^N)$	20
Trellis Marginals	$\mathcal{O}(3^{N-1})$	$\mathcal{O}(2^N)$	20
Sparse Trellis Partition Function	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert^{log(3)})$	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert)$	100–1000
Sparse Trellis MAP Clustering	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert^{log(3)})$	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert)$	100–1000
Sparse Trellis Marginals	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert^{log(3)})$	$\mathcal{O}(\lvert\widehat{\mathbb{T}}\rvert)$	100–1000
Slab Tree Construction	$\mathcal{O}(N)$	$\mathcal{O}(N)$	10000+
Slab Tree Probability Query	$\mathcal{O}(Nlog(N))$	$\mathcal{O}(N)$	10000+

helpfully reduce the state space considered and represent distributions over clusterings that do not violate some fundamental assumptions in the application domain. Several recent papers (see [16], [39], [37] for examples) have used constrained clustering in applications requiring some specified measures of fairness. There are several ways one might specify clustering constraints. The classic approach in flat clustering is pairwise instance-based constraints (i.e., constraints on whether two elements in X should/should not be clustered together, referred to as must-link/cannot-link, respectively) [6], which can be expressed with certain equivalent geometric distance constraints (where elements within ϵ/δ must/must not be clustered together) [26]. Other forms of constraints in flat clustering have also been explored, including cluster size constraints [6], and balancing with respect to an existing clustering [16]. Flat clustering constraints are not directly applicable to hierarchical clustering, and there has been work developing constraints for hierarchical clustering as well, with triplet constraints (i.e., that two elements in X must be clustered earlier in the hierarchy than with a third element)[72] being common [15, 14, 13].

The trellis can effectively represent several forms of constraints, including pairwise instance-based constraints, triplet constraints, and balance constraints.

The must-link constraint, where (x_i,x_j) must be in the same cluster corresponds to a trellis where the leaves corresponding to elements x_i and x_j are merged together to form a new leaf, x_{ij}. A trellis with a single instance of this constraint would then be sized $2^{n-1} - 1$ rather than $2^n - 1$, and in general applying k constraints of this type reduces the trellis size to $2^{n-k} - 1$.

The cannot-link constraint, where (x_i,x_j) can't be clustered together corresponds to a trellis where all the clusters containing (x_i,x_j) are removed (sans the root). As above, a trellis with a single instance of this constraint would then be sized $2^{n-1} - 1$ rather than $2^n - 1$. In the general case, there is the possibility between overlap between the pairs being removed, so the trellis size is a function of the number of these constraints along with the overlap among constraints.

Triplet constraints, where (x_i,x_j) must be joined together before either is joined with x, corresponds to a trellis where there are no nodes containing x_i and x (or x_j and x) that do not also contain x_j (or x_i).

Interestingly, we're able to easily extend triplet constraints to more a general kind of constraint, i.e., $(x_a, x_b, x_c|x_x, x_y, x_z)$, indicating[2] that x_a, x_b, and x_c, must be merged prior to being merged with x_x and x_y and x_z. In this example, x_a and x_x can to be joined together at the leaves, but then joining x_a and x_x with x_y and x_z would violate the constraint. Note that constraints of this type correspond to a trellis where x_a (or x_b or x_c) does not appear in a cluster with x_x, x_y, and x_z without also appearing with x_b and x_c (etc.).

It is important to note that all of the above approaches encode *hard* constraints, meaning that the constraint set must be satisfiable for there to be a solution (i.e.,

[2]Note that swapping "or" in place of "and" here makes $(x_a, x_b|x_x, x_y, x_z)$ corresponds to $(x_a, x_b|x_x)$ + $(x_a, x_b|x_y)$ + $(x_a, x_b|x_z)$).

solutions meeting only some of the constraints are never considered). It is also possible apply soft constraints by reducing cluster potential functions rather than removing them from the trellis, and in this way both soft and hard constraints can be encoded.

Balance constraints in clustering have been proposed in the context of fairness where membership in a demographic is an auxiliary clustering, and the measure of balance within a cluster is minimum ratio of demographics within a cluster, though this type of constraint can be viewed generally as clustering with respect to an existing partition. When using hard constraints, a balance threshold is chosen in order to include/exclude certain clusters, and soft constraints can be used by increasing/decreasing each clusters potential according to the balance.

Despite the trellis naturally encodes a variety of important constraints, it struggles to effectively encode constraints based on the cluster sizes. For example, if attempting to encode the constraint that all clusters must either be roughly the same size, it would be necessary to include both large and small clusters (since clusterings that consist of few large clusters or many small clusters both satisfy the constraint). However, the trellis would also include clusterings consisting of both small and large clusters, which fail the constraint.

7.2 Algorithms for Arbitrarily-Shaped Flat Sparse Trellises

The flat clustering algorithms using sparse trellises are only able to be run on sparse trellises closed under recursive complement or are tree-structured. It would be generally useful to be able to compute the partition function and MAP given arbitrarily-shaped trellises without having to close them under recursive complement.

One relaxation that could be considered is whether a cluster forms any valid k-ary partition under a descendant, rather than a complement (i.e., a two-partition). This represents a much broader set of trellises and could prove to be useful in practice.

However, it would be necessary to find the com (perhaps represented as edges in the trellis), and to carefully consider how to account for k in the algorithms.

Another direction would be to find or establish positive results regarding relationship between MAP hierarchy and MAP flat-clustering and/or between the hierarchical clustering partition function and flat clustering partition function, which broaden the set of flat clustering trellises to those usable for hierarchical clustering.

Otherwise, the primary challenge in computation using sparse flat clustering trellises is how to deal w/intersections of elements in trellis nodes (since we can't simply consider the partition of a node with its complement in the trellis). One potentially productive direction would be to consider methods from sieve theory.

7.3 Flat and Hierarchical Sparse Trellis Growing Techniques

We described some sparse trellis construction techniques in Sections 4.1.1.4 and 6.1.4, though many more approaches exist.

For example, one can start with the leaves (or root), adding (removing) a single element to each of the leaves (the root). If the potential increases, add it to the sparse trellis, otherwise do not. Iterate, continuing to add (remove) one element from each cluster added to the sparse trellis in the previous iteration, and adding to the sparse trellis if the resultant cluster increases in potential. In flat clustering, whenever the potential function is monotonically increasing along some path[3], where the steps are single joins (splits) from the shattered (single) partition to the MAP clustering, a sparse trellis grown in this way is guaranteed contain the MAP clustering. While in the worst case this could build a full trellis, this seems like a promising and exciting direction for empirical exploration, since in the best case this makes finding the exact MAP tractable for relatively large datasets.

[3]This is the case for correlation clustering and, we expect, many other clustering settings.

Since sparse trellises are representations of distributions over clusterings, it would be interesting to treat sparse trellis construction as a representation learning problem. One concrete idea in this direction involves using the full trellis as the structure of a neural network and learning the sparse trellis encoding (essentially a sparsification approach (see Section 4.1.1.4)). Another direction to consider is, given a set of high/low potential clusterings, learning the potential function between pairs of elements in the dataset.

In Section 7.5 we describe using trellises as search structures, as well as how to search without creating trellises in advance. As a result, various search techniques create sparse trellises a by-product.

7.4 Building on Slab Trees

Section 4.1.2 covers only the most fundamental ideas in slab trees, and there are many directions one could go to expand and improve upon Slab trees. We list just a few here:

- *Multiple slab trees*, rather than just a single tree, are able to represent more and more complex distributions.
- *Slab sparse trellises*, similar to multiple slab trees, represent more and more complex distributions than using a tree.
- *Marginals*, can be computed according to slab trees using methods similar to those described in Sections 5.1.4.
- *Slab trees for hierarchical clusterings* are possible as well, where the slab values are given to hierarchies in a manner nearly identical to flat clusterings.
- *Jointly building the slab tree and setting the slab values* should result in slab trees that better represent the exact distribution.

Among other things, can be used to build sparse trellises for computing Z. Similar to the sparse trellis construction techniques discussed in 7.3, the search techniques described in 7.5 could provide a way to set slab values, as an alternative to using existing partition function estimation techniques.

7.5 Trellises as Search Structures

Cost functions such as Dasgupta's cost enables hierarchical clustering to be cast as an optimization problem. Similarly, ad hoc methods for hierarchical clustering (e.g., greedy agglomerative, divisive, etc.) can be viewed as search over the space we are of all hierarchical clusterings. Viewing clustering as search leads naturally to the use of existing search methods, some which have seen little to no use for clustering, since applying search naively leads to a super-exponentially large space and time complexity. The trellis data structure, as a compact encoding of search state space as paths in the trellis, and associated dynamic programming techniques hold great promise for approaching clustering as search.

Further, searching over sparse trellises yields a natural approximation under an augmented trellis data structure that enables the algorithm to scale to larger data sets. One is able to search over the clusterings represented in the sparse trellis. It is also possible to search a broader space of clusterings, using the sparse trellis as a search space initializer, with the missing vertices and edges representing states not yet explored, and the search extending from existing vertices. Among other approaches, we are hopeful we might be able to bring graph streaming techniques to bear in cases where we are searching large scale graphs (e.g., by treating the two-partitions at each node as coming from a data stream).

We find this a very exciting direction, with many possible additional spin-off directions, and we are actively researching clustering as search.

BIBLIOGRAPHY

[1] Adams, Ryan P. Hierarchical agglomerative clustering.

[2] Aldous, David J. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*. Springer, 1985, pp. 1–198.

[3] Antoniak, Charles E. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics* (1974), 1152–1174.

[4] Arora, Sanjeev, Lee, James, and Naor, Assaf. Euclidean distortion and the sparsest cut. *Journal of the American Mathematical Society 21*, 1 (2008), 1–21.

[5] Bansal, Nikhil, Blum, Avrim, and Chawla, Shuchi. Correlation clustering. *Machine learning 56*, 1-3 (2004), 89–113.

[6] Basu, Sugato, Davidson, Ian, and Wagstaff, Kiri. *Constrained clustering: Advances in algorithms, theory, and applications*. CRC Press, 2008.

[7] Blei, David M, Griffiths, Thomas L, and Jordan, Michael I. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)* (2010).

[8] Blundell, C, Teh, YW, and Heller, KA. Bayesian rose trees. In *Uncertainty in Artificial Intelligence, (UAI)* (2010).

[9] Boyles, Levi, and Welling, Max. The time-marginalized coalescent prior for hierarchical clustering. In *Advances in Neural Information Processing Systems* (2012), pp. 2969–2977.

[10] Cacciari, Matteo, Salam, Gavin P., and Soyez, Gregory. The anti-k_t jet clustering algorithm. *JHEP 04* (2008), 063.

[11] Callan, David. A combinatorial survey of identities for the double factorial, 2009.

[12] Charikar, Moses, and Chatziafratis, Vaggos. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (2017), SIAM, pp. 841–854.

[13] Charikar, Moses, Chatziafratis, Vaggos, and Niazadeh, Rad. Hierarchical clustering better than average-linkage. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (2019), SIAM, pp. 2291–2304.

[14] Charikar, Moses, Chatziafratis, Vaggos, Niazadeh, Rad, and Yaroslavtsev, Grigory. Hierarchical clustering for euclidean data. In *The 22nd International Conference on Artificial Intelligence and Statistics* (2019), PMLR, pp. 2721–2730.

[15] Chatziafratis, Vaggos, Niazadeh, Rad, and Charikar, Moses. Hierarchical clustering with structural constraints. *arXiv preprint arXiv:1805.09476* (2018).

[16] Chierichetti, Flavio, Kumar, Ravi, Lattanzi, Silvio, and Vassilvitskii, Sergei. Fair clustering through fairlets. In *Advances in Neural Information Processing Systems* (2017), pp. 5029–5037.

[17] Cimiano, Philipp, and Staab, Steffen. Learning concept hierarchies from text with a guided agglomerative clustering algorithm. In *Proceedings of the ICML 2005 Workshop on Learning and Extending Lexical Ontologies with Machine Learning Methods* (2005).

[18] Cohen-Addad, Vincent, Kanade, Varun, and Mallmann-Trenn, Frederik. Hierarchical clustering beyond the worst-case. In *Advances in Neural Information Processing Systems (NeurIPS)* (2017).

[19] Cohen-Addad, Vincent, Kanade, Varun, Mallmann-Trenn, Frederik, and Mathieu, Claire. Hierarchical clustering: Objective functions and algorithms. *Journal of the ACM (JACM) 66*, 4 (2019), 1–42.

[20] Cranmer, Kyle, Macaluso, Sebastian, and Pappadopulo, Duccio. Toy Generative Model for Jets, 2019. Toy Generative Model for Jets.

[21] Cranmer, Kyle, Macaluso, Sebastian, and Pappadopulo, Duccio. Toy Generative Model for Jets Package, 2019. https://github.com/SebastianMacaluso/ToyJetsShower.

[22] Cranmer, Kyle, Macaluso, Sebastian, and Pappadopulo, Duccio. Generation and Inference Unification in Jet Physics, 2020. Work in progress.

[23] Dale, E., and Moon, J. The permuted analogues of three Catalan sets, 1993.

[24] Dasgupta, Sanjoy. A cost function for similarity-based hierarchical clustering. In *Symposium on Theory of Computing (STOC)* (2016).

[25] Dasgupta, Sanjoy. A cost function for similarity-based hierarchical clustering. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016* (2016), pp. 118–127.

[26] Davidson, Ian, and Ravi, SS. The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data mining and knowledge discovery 14*, 1 (2007), 25–61.

[27] Dechter, Rina. Bucket elimination: A unifying framework for probabilistic inference. 1999.

[28] Dheeru, Dua, and Karra Taniskidou, Efi. UCI machine learning repository, 2017.

[29] Geman, Stuart, and Geman, Donald. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence* (1984).

[30] Ghahramani, Zoubin, Jordan, Michael I, and Adams, Ryan P. Tree-structured stick breaking for hierarchical data. In *Advances in neural information processing systems* (2010), pp. 19–27.

[31] Greenberg, Craig, Monath, Nicholas, Kobren, Ari, Flaherty, Patrick, McGregor, Andrew, and McCallum, Andrew. Compact representation of uncertainty in clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[32] Greenberg, Craig, Monath, Nicholas, Kobren, Ari, Flaherty, Patrick, McGregor, Andrew, and McCallum, Andrew. Compact representation of uncertainty in clustering. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 8630–8640. http://papers.nips.cc/paper/8081-compact-representation-of-uncertainty-in-clustering.pdf.

[33] Heller, Katherine A, and Ghahramani, Zoubin. Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning* (2005), pp. 297–304.

[34] Hu, Yuening, Ying, Jordan L, Daume III, Hal, and Ying, Z Irene. Binary to bushy: Bayesian hierarchical clustering with the beta coalescent. In *Advances in Neural Information Processing Systems (NeurIPS)* (2013).

[35] Hubert, Lawrence, Arabie, Phipps, and Meulman, Jacqueline. *Combinatorial data analysis: Optimization by dynamic programming*. Society for Industrial and Applied Mathematics, 2001.

[36] Jensen, Robert E. A dynamic programming algorithm for cluster analysis. *Operations Research* (1969).

[37] Jing, Imtiaz Masud Ziko Eric Granger, and Ayed, Yuan Ismail Ben. Clustering with fairness constraints: A flexible and scalable approach.

[38] Kappes, Jörg Hendrik, Swoboda, Paul, Savchynskyy, Bogdan, Hazan, Tamir, and Schnörr, Christoph. Probabilistic correlation clustering and image partitioning using perturbed multicuts. In *International Conference on Scale Space and Variational Methods in Computer Vision* (2015).

[39] Kleindessner, Matthäus, Samadi, Samira, Awasthi, Pranjal, and Morgenstern, Jamie. Guarantees for spectral clustering with fairness constraints. *arXiv preprint arXiv:1901.08668* (2019).

[40] Knowles, David A, and Ghahramani, Zoubin. Pitman-yor diffusion trees. In *Conference on Uncertainty in Artificial Intelligence (UAI)* (2011).

[41] Kobren, Ari, Monath, Nicholas, Krishnamurthy, Akshay, and McCallum, Andrew. An online hierarchical algorithm for extreme clustering. *arXiv preprint arXiv:1704.01858* (2017).

[42] Kohonen, Jukka, and Corander, Jukka. Computing exact clustering posteriors with subset convolution. *Communications in Statistics-Theory and Methods* (2016).

[43] Kolmogorov, AN. Foundations of the theory of probability.

[44] Kraskov, Alexander, Stögbauer, Harald, Andrzejak, Ralph G, and Grassberger, Peter. Hierarchical clustering using mutual information. *EPL (Europhysics Letters) 70*, 2 (2005), 278.

[45] Lehmann, Brian D, and Pietenpol, Jennifer A. Identification and use of biomarkers in treatment strategies for triple-negative breast cancer subtypes. *The Journal of pathology* (2014).

[46] Louppe, Gilles, Cho, Kyunghyun, Becot, Cyril, and Cranmer, Kyle. QCD-Aware Recursive Neural Networks for Jet Physics. *JHEP 01* (2019), 057.

[47] Lovász, László. Combinatorial problems and exercises.

[48] Ma, Jianzhu, Peng, Jian, Wang, Sheng, and Xu, Jinbo. Estimating the partition function of graphical models using langevin importance sampling. In *Artificial Intelligence and Statistics* (2013), pp. 433–441.

[49] Matula, David W, and Shahrokhi, Farhad. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics 27*, 1-2 (1990), 113–123.

[50] Mises, R von. Vorlesungen aus dem gebiete der angewandte mathematik. i. band. wahrscheinlichkeitsrechnung. *F. Deuticke, Leipzig und Wien* (1931).

[51] Mitchell, Toby J, and Beauchamp, John J. Bayesian variable selection in linear regression. *Journal of the American Statistical Association 83*, 404 (1988), 1023–1032.

[52] Molkaraie, Mehdi, and Loeliger, Hans-Andrea. Monte carlo algorithms for the partition function and information rates of two-dimensional channels. *IEEE Transactions on Information Theory 59*, 1 (2012), 495–503.

[53] Moseley, Benjamin, and Wang, Joshua. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Advances in Neural Information Processing Systems* (2017).

[54] Neal, Radford M. Density modeling and clustering using dirichlet diffusion trees. *Bayesian statistics* (2003).

[55] Network, Cancer Genome Atlas, et al. Comprehensive molecular portraits of human breast tumours. *Nature* (2012).

[56] Paisley, John, Wang, Chong, Blei, David M, and Jordan, Michael I. Nested hierarchical dirichlet processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014).

[57] Papandreou, George, and Yuille, Alan L. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *2011 International Conference on Computer Vision* (2011), IEEE, pp. 193–200.

[58] Pitman, Jim, et al. Combinatorial stochastic processes. Tech. rep., Technical Report 621, Dept. Statistics, UC Berkeley, 2002. Lecture notes for ..., 2002.

[59] Potamianos, Gerasimos, and Goutsias, John. Stochastic approximation algorithms for partition function estimation of gibbs random fields. *IEEE Transactions on Information Theory 43*, 6 (1997), 1948–1965.

[60] Reddy, D Raj, et al. Speech understanding systems: A summary of results of the five-year research effort. department of computer science, 1977.

[61] Rouzier, Roman, Perou, Charles M, Symmans, W Fraser, et al. Breast cancer molecular subtypes respond differently to preoperative chemotherapy. *Clinical cancer research* (2005).

[62] Roy, Aurko, and Pokutta, Sebastian. Hierarchical clustering via spreading metrics. *The Journal of Machine Learning Research 18*, 1 (2017), 3077–3111.

[63] Saddiki, Hachem, McAuliffe, Jon, and Flaherty, Patrick. Glad: a mixed-membership model for heterogeneous tumor subtype classification. *Bioinformatics* (2014).

[64] Singh, Sameer, Subramanya, Amarnag, Pereira, Fernando, and McCallum, Andrew. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1* (2011), Association for Computational Linguistics, pp. 793–803.

[65] Sørlie, Therese, Perou, Charles M, Tibshirani, Robert, Aas, Turid, Geisler, Stephanie, Johnsen, Hilde, Hastie, Trevor, Eisen, Michael B, Van De Rijn, Matt, Jeffrey, Stefanie S, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences 98*, 19 (2001), 10869–10874.

[66] Sørlie, Therese, Tibshirani, Robert, Parker, Joel, et al. Repeated observation of breast tumor subtypes in independent gene expression data sets. *Proceedings of the National Academy of Sciences* (2003).

[67] Suchard, Marc A, Lemey, Philippe, Baele, Guy, Ayres, Daniel L, Drummond, Alexei J, and Rambaut, Andrew. Bayesian phylogenetic and phylodynamic data integration using beast 1.10. *Virus evolution* (2018).

[68] Suzuki, Ryota, and Shimodaira, Hidetoshi. Pvclust: an r package for assessing the uncertainty in hierarchical clustering. *Bioinformatics 22*, 12 (2006), 1540–1542.

[69] Teh, Yee W, Daume III, Hal, and Roy, Daniel M. Bayesian agglomerative clustering with coalescents. In *Advances in Neural Information Processing Systems (NeurIPS)* (2008).

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

[71] Van Os, BJ, and Meulman, Jacqueline J. Improving dynamic programming strategies for partitioning. *Journal of classification 21*, 2 (2004), 207–230.

[72] Vikram, Sharad, and Dasgupta, Sanjoy. Interactive bayesian hierarchical clustering. In *International Conference on Machine Learning* (2016), pp. 2081–2090.

[73] Vogt, Henri. *Leçons sur la résolution algébrique des équations*. Nony, 1895.

[74] Wainwright, Martin J, Jaakkola, Tommi S, and Willsky, Alan S. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory 51*, 7 (2005), 2313–2335.

[75] Wang, Liangliang, Bouchard-Côté, Alexandre, and Doucet, Arnaud. Bayesian phylogenetic inference using a combinatorial sequential monte carlo method. *Journal of the American Statistical Association 110*, 512 (2015), 1362–1374.

[76] Wick, Michael, Singh, Sameer, and McCallum, Andrew. A discriminative hierarchical model for fast coreference at large scale. In *Association for Computational Linguistics (ACL)* (2012).

[77] Yersal, Ozlem, and Barutca, Sabri. Biological subtypes of breast cancer: Prognostic and therapeutic implications. *World journal of clinical oncology* (2014).

[78] Zanella, Giacomo, Betancourt, Brenda, Wallach, Hanna, Miller, Jeffrey, Zaidi, Abbas, and Steorts, Rebecca C. Flexible models for microclustering with application to entity resolution. *Advances in Neural Information Processing Systems* (2016).

[79] Zhang, Nevin Lianwen, and Poole, David. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research* (1996).

[80] Zhang, Yutao, Zhang, Fanjin, Yao, Peiran, and Tang, Jie. Name disambiguation in aminer: Clustering, maintenance, and human in the loop. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2018), ACM, pp. 1002–1011.