Exact and Approximate Hierarchical Clustering Using A*

Craig S. Greenberg*,1, Sebastian Macaluso*,2, Nicholas Monath*,3,
Avinava Dubey4, Patrick Flaherty5, Manzil Zaheer4, Amr Ahmed4,
Kyle Cranmer2, and Andrew McCallum3

1National Institute of Standards and Technology, USA
2Center for Cosmology and Particle Physics & Center for Data Science, New York University, USA
3College of Information and Computer Sciences, University of Massachusetts Amherst, USA
4Google Research, Mountain View, CA, USA
5Department of Mathematics and Statistics, University of Massachusetts Amherst, USA

Abstract

Hierarchical clustering is a critical task in numerous domains. Many approaches are based on heuristics and the properties of the resulting clusterings are studied post hoc. However, in several applications, there is a natural cost function that can be used to characterize the quality of the clustering. In those cases, hierarchical clustering can be seen as a combinatorial optimization problem. To that end, we introduce a new approach based on A* search. We overcome the prohibitively large search space by combining A* with a novel *trellis* data structure. This combination results in an exact algorithm that scales beyond previous state of the art, from a search space with 10^{12} trees to 10^{15} trees, and an approximate algorithm that improves over baselines, even in enormous search spaces that contain more than 10^{1000} trees. We empirically demonstrate that our method achieves substantially higher quality results than baselines for a particle physics use case and other clustering benchmarks. We describe how our method provides significantly improved theoretical bounds on the time and space complexity of A* for clustering.

1 Introduction

Hierarchical clustering has been applied in a wide variety of settings such as scientific discovery [1], personalization [2], entity resolution for knowledge-bases [3, 4, 5], and jet physics [6, 7, 8, 9]. While much work has focused on approximation methods for relatively large datasets [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20], there are also important use cases of hierarchical clustering that demand exact or high-quality approximations [21]. This
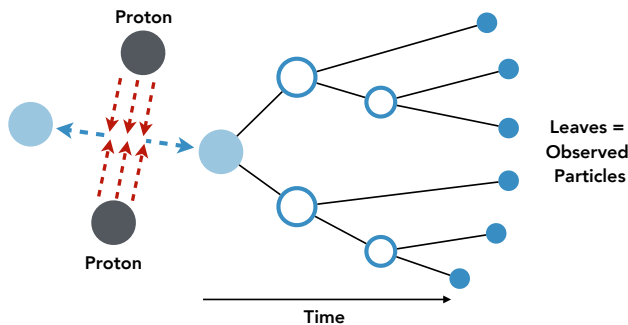
---

*The three authors contributed equally to this work.

Figure 1: **Jets as hierarchical clustering**. Schematic representation of a process producing a jet at the Large Hadron Collider at CERN. Constituents of the incoming protons interact and produce two new particles (solid light blue). Each of them goes through successive binary splittings until giving rise to stable final state particles (solid dark blue), which hit the detector. These final state particles (leaves of a binary tree) form a jet, and each possible hierarchical clustering represents a different splitting history.

paper focuses on one such application of hierarchical clustering in jet physics: inferring *jet structures*.

Jet Physics: Particle collisions in collider experiments, such as the Large Hadron Collider (LHC) at CERN, produce new particles that go through a series of successive binary splittings, termed a *showering process*, which finally leads to a *jet*: a collimated set of stable-charged and neutral particles that are measured by a detector. A schematic representation of this process is shown in Figure 1, where the jet constituents are the (observed) leaf particles in solid dark blue and the intermediate (latent) particles can be identified as internal nodes of the hierarchical clustering. There has been a significant amount of research into jet clustering over the decades, i.e. reconstructing the showering process (hierarchical clustering) from the observed jet constituents (leaves). The performance of these algorithms is often the bottleneck to precision measurements of the Standard Model of particle physics and searches for new physics. The LHC generates enormous datasets, but for each recorded collision there is a hierarchical clustering task with typically 10 to 100 particles (leaves). Despite the relatively small number of elements in these applications, exhaustive solutions are intractable, and current exact methods, e.g., [21], have limited scalability. The industry standard uses agglomerative clustering techniques, which are greedy [22]. Thus, they tend to find low-likelihood hierarchical clusterings, which results in a poor inference of the properties of the initial state particles (solid light blue circles in Figure 1). As a result, their type could be misidentified (e.g., top quark, W boson, gluon, etc.). Also, the success of deep learning jet classifiers [23] (for these initial state particles) that do not use clustering information is evidence for the deficiencies of current clustering algorithms since the *right* clustering would mean that traditional clustering-based physics observables would be essentially optimal in terms of classification performance. Thus, exact and high-quality approximations in this

2

context would be highly beneficial for data analyses in experimental particle physics.

Traditional algorithms, such as greedy approximation methods can converge to local optima, and have little ability to re-consider possible alternatives encountered at previous steps in their search for the optimal clustering. This naturally raises the question of how well-established, *non-greedy* search algorithms, such as A*, can be applied to hierarchical clustering.

An immediate challenge in using A* to search for the optimal tree structure for a given objective is the vast size of the space of hierarchical clusterings. There are *many* possible hierarchical clusterings for n elements, specifically $(2n - 3)!!$, where !! indicates double factorial. A naïve application of A* would require super-exponential time and space. Indeed, the only previous work exploring the use of A* for clustering, [24], uses A* to find MAP solutions to Dirichlet Process Mixture Models and has no bounds on the time and space complexity of the algorithm.

1.1 Theoretical contributions of this paper

- **A* Datastructure**. Inspired by [25, 21], we present a data structure to compactly encode the state space and objective value of hierarchical clusterings for search, using a sequence of nested min-heaps (§3.3).

- **Time & Space Bounds**. Using this structure, we are able to bound the running time and space complexity of using A* to search for clusterings, an improvement over previous work (§3.3).

1.2 Empirical contributions of this paper

- **Jet Physics**. We also demonstrate empirically that A* can find exact solutions to larger jet physics datasets than previous work [21], and can improve over benchmarks among approximate methods in data sets with enormous search spaces (exceeding 10^{300} trees). (§5).

- **Clustering benchmarks**. We find that A* search finds solutions with improved hierarchical correlation clustering cost compared with common greedy methods on benchmark datasets [26].(§6).

2 Preliminaries

A hierarchical clustering is a recursive partitioning of a dataset into nested subsets:

Definition 1. (Hierarchical Clustering) *Given a dataset of elements, $X = \{x_i\}_{i=1}^{N}$, a **hierarchical clustering**, H, is a set of nested subsets of X, s.t. $X \in H$, $\{\{x_i\}\}_{i=1}^{N} \subset H$, and $\forall X_i, X_j \in H$, either $X_i \subset X_j$, $X_j \subset X_i$, or $X_i \bigcap X_j = \emptyset$. Further, $\forall X_i \in H$, if $\exists X_j \in H$ s.t. $X_j \subset X_i$, then $\exists X_k \in H$ s.t. $X_j \bigcup X_k = X_i$.*
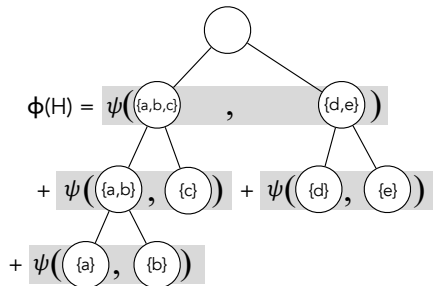
Figure 2: **Family of Hierarchical Clustering Cost Functions**. This paper studies cost functions that decompose as the sum of a cost term computed for each pair of sibling nodes in the tree structure.

Consistent with our application in jet physics and previous work [21], we limit our consideration to hierarchical clusterings with binary branching factor.[1]

Hierarchical clustering cost functions represent the quality of a hierarchical clustering for a particular dataset. In this work, we study a general family of hierarchical clustering costs defined as the sum over pairs of sibling clusters in the structure (Figure 2). This family of costs generalizes well known costs like Dasgupta's [28]. Formally, the family of hierarchical clustering costs we study in this paper can be written as:

Definition 2. (Hierarchical Clustering Costs) *Let X be a dataset, H be a hierarchical clustering of X, let $\psi : 2^X \times 2^X \to \mathbb{R}^+$ be a function describing the cost incurred by a pair of sibling nodes in H. We define the cost, $\phi(H)$ of a hierarchical clustering H as:*

$$\phi(H) = \sum_{X_L, X_R \in \textbf{\textit{sibs}}(H)} \psi(X_L, X_R) \tag{1}$$

where $\textbf{\textit{sibs}}(H) = \{(X_L, X_R) | X_L \in H, X_R \in H, X_L \cap X_R = \emptyset, X_L \cup X_R \in H\}$.

The goal of hierarchical clustering is then to find the lowest cost H among all hierarchical clusterings of X, $\mathbb{H}(X)$, i.e., $\operatorname{argmin}_{H \in \mathbb{H}(X)} \phi(H)$.

3 A* Search for Clustering

The fundamental challenge of applying A* to clustering is the massive state space. Naïve representations of the A* state space and frontier require explicitly storing every tree structure, potentially growing to be at least the number of binary trees $(2n - 3)!!$. To overcome this, we propose an approach using a *cluster trellis* to (1) compactly encode

---

[1]Binary trees encode at least as many *tree consistent partitions* as multifurcating / n-ary trees [27, 21] and for well known cost functions, such as Dasgupta's cost, provide trees of lower cost than multifurcating / n-ary trees [28].

states in the space of hierarchical clusterings (as paths from the root to the leaves of the trellis), and (2) compactly represent the search frontier (as nested priority queues). We first describe how the cluster trellis represent the A* state space and frontier, and then show how A* search this data structure to find the lowest cost clustering.

3.1   A* Search and State Space

A* is a best-first search algorithm for finding the minimum cost path between a starting node and a goal node in a weighted graph. Following canonical notation, the function $f(n) = g(n) + h(n)$ determines the "best" node to search next, where $g(n)$ is the cost of the path up from the start to node n and $h(n)$ is a heuristic estimating the cost of traveling from n to a goal.

When the heuristic $h(\cdot)$ is *admissible* (under estimates cost), A* is admissible and provides an optimal solution. If $h(\cdot)$ is both admissible and *consistent* or *monotone* (the heuristic cost estimate of reaching a goal from state x, $h(x)$, is less than the actual cost of traveling from state x to state y plus the heuristic cost estimate of reaching a goal from state y, $h(y)$), then A* is *optimally efficient*[2].

In order to use A* to search for clusterings, we need to define the search space, the graph being searched over, and the goal states.

Definition 3. (**Search State / Partial Hierarchical Clustering**) *We define each state in the search space to be a **partial hierarchical clustering**, $\breve{H}$, which is a subset of some hierarchical clustering of the dataset X, i.e., $\exists H \in \mathbb{H}(X)$, s.t. $\breve{H} \subseteq H$. A state is a **goal** state if it is a hierarchical clustering, e.g. $\breve{H} \in \mathbb{H}(X)$.*

3.2   The Cluster Trellis for Compactly Encoding the Search Space

It is intractable to search over a graph that naïvely encodes these states, even when using A*, as the number of goal states is massively large $((2N-3)!!)$, and therefore the total number of states is even larger. Such a naïve search could require super exponential space to maintain the the frontier, i.e., the set of states A* is currently open to exploring. We propose an alternative encoding that utilizes a trellis data structure to encode the frontier and state space. Our proposed encoding reduces the super-exponential space and time required for A* to find the exact MAP to exponential in the worst case.

Previous work, [21], defines the *cluster trellis for hierarchical clustering* (hereafter denoted *trellis*) as a directed acyclic graph, $\mathbb{T}$, where, like a hierarchical clustering, the leaves of the trellis, $\mathsf{lvs}(\mathbb{T})$, correspond to data points. The internal nodes of $\mathbb{T}$ correspond to

---

[2]It is worth noting that there is a trade-off between the quality of the heuristic and the number of iterations of A*, with better heuristics potentially resulting in fewer iterations at the cost of taking longer to compute. An extreme example being that a perfect heuristic will require no-backtracking, except in cases of ties.
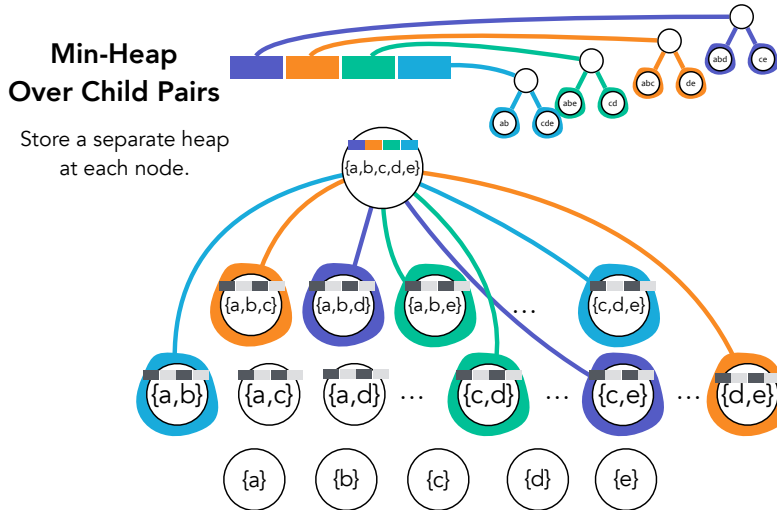
Figure 3: **Nested Min Heaps** The A* search space is compactly encoded in the trellis. Each node stores a min-heap ordering pairs of its children. Each pair encodes a two partition (e.g. split) of its parents points.

clusters, and edges in the trellis are defined between child and parent nodes, where children contain a subset of the points contained by their parents.

Definition 4. *(Cluster Trellis for Hierarchical Clustering) [21] Given a dataset of elements, $X = \{x_i\}_{i=1}^N$, a trellis $\mathbb{T}$ is a directed acyclic graph with vertices corresponding to subsets of X, i.e., $\mathbb{T} = \{T_1, \ldots, T_k\} \subseteq \mathbb{P}(X)$, where $\mathbb{P}(X)$ is the powerset of X. Edges in the trellis are defined between a child node $C \in \mathbb{T}$ and a parent $P \in \mathbb{T}$ such that $P \setminus C \in \mathbb{T}$ (i.e., C and $P \setminus C$ form a two-partition of P).*

Note that the trellis is a graph that compactly represents tree structures, enabling the use of various graph search algorithms for hierarchical clustering.

3.3 A* on the Cluster Trellis

In this section, we describe how to use of the trellis to run A* in less than super-exponential time and space. We show that when using a 'full' trellis, one that contains all possible subsets of the dataset (i.e., $\mathbb{T} = \mathbb{P}(X)$), A* will find an exact solution, i.e., the optimal (minimum possible cost) hierarchical clustering. Interestingly, when the trellis is "sparse", i.e., contains only a subset of all possible subsets ($\mathbb{T} \subset \mathbb{P}(X)$), we further show that A* finds the lowest cost hierarchical clustering among those encoded by the trellis structure. In our proceeding analysis, we prove the correctness as well as the time and space complexity of running A* search on a trellis.

6

To run A* on trellis $\mathbb{T}$, we store at each node, X_i, in $\mathbb{T}$ a min heap over two-partition pairs, which correspond to the node's children, Π, which we denote as $X_i[\Pi]$. Each node also stores the value of the lowest cost split at each node, $X[\Phi]$, as well as a pointer to children associated with the best split, $X[\Xi] = \{X_\ell, X_r\}$.

The starting state of the search refers to the root node corresponding to the full dataset, X. We compactly encode a partial hierarchical clustering as a set of paths through the trellis structure. In particular,

$$H_\Xi(X) = \{X\} \bigcup_{X_c \in X[\Xi]} H_\Xi(X_c) \tag{2}$$

where $X[\Xi] = \emptyset$ if $X \in \mathsf{lvs}(H)$, where $\mathsf{lvs}(H) = \{X \mid X \in H, \nexists X' \in H, X' \subset X\}$. The partial hierarchical clustering can be thought of as starting at the root node, adding to the tree structure the pair of children at the top of the root node's min heap, and then for each child in this pair, selecting the pair of children (grandchildren of the root) from their min-heaps to add to the tree structure. The algorithm proceeds in this way, adding descendants until it reachs nodes that do not yet have min-heaps initialized.

Each node's min heap, $X_i[\Pi]$, stores five-element tuples:

$$(f_{LR}, \ g_{LR}, \ h_{LR}, X_L, \ X_R) \in X_i[\Pi] \tag{3}$$

Exploration in A* consists of instantiating the min heap of a given node, where the heap ordering is determined by $f_{LR} = g_{LR} + h_{LR}$. The value g_{LR} corresponds to the g value of the partial hierarchical clustering rooted at $X_L \cup X_R$, including both the nodes X_L and X_R. Formally:

$$g_{LR} = \psi(X_L, X_R) + \sum_{X_{LL}, X_{LR} \in \mathsf{sibs}(H_\Xi(X_L))} \psi(X_{LL}, X_{LR}) + \sum_{X_{RL}, X_{RR} \in \mathsf{sibs}(H_\Xi(X_R))} \psi(X_{RL}, X_{RR}) \tag{4}$$

Recall that H_Ξ represents a complete or partial hierarchical clustering. The value h gives an estimate of the potential for all of the leaves of H_Ξ, and is defined to be 0 if all of the leaf nodes of H_Ξ are singleton clusters (in which case H_Ξ is a complete hierarchical clustering). Formally:

$$h_{LR} = \sum_{X_\ell \in \mathsf{lvs}(H_\Xi(X_L \cup X_R))} \mathbb{H}(X_\ell) \tag{5}$$

where $\mathbb{H}(X_\ell)$ is the objective-specific heuristic function required by A*, and $\mathbb{H}(X_\ell) = 0$ if X_ℓ is a singleton.

Given these definitions, A* can be executed in the following way. First, follow the paths from the root of the trellis to find the best partial hierarchical clustering (using Eq 2). If this state is a goal state and the h_{LR} value at the top of the min heap is 0, then return

Partial Hierarchical Clustering State

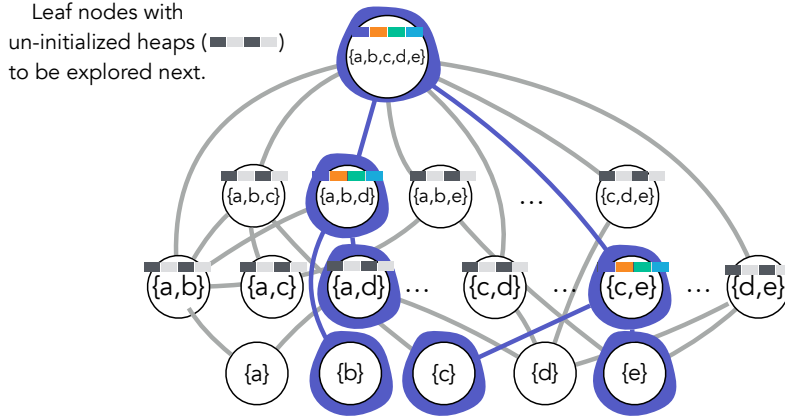Leaf nodes with un-initialized heaps (▪▬▪▬▪) to be explored next.

Figure 4: **State Space** A state (i.e., a partial hierarchical clustering) is compactly encoded as paths in the trellis structured, discovered by following the root-to-leaf pointers.

this structure. If the state is not a goal state, explore the leaves of the partial hierarchical clustering, instantiating the min heap of each leaf with the children of the leaf nodes and their corresponding f_{LR}, g_{LR}, h_{LR} values (from Eq. 4 & 5). Then, after exploration, update the heap values of g_{LR} and h_{LR} in each of the ancestor nodes (in the partial hierarchical clustering) of the newly explored leaves. See Algorithm 1 for a summary in pseudocode.

First, we show that Algorithm 1 will find the exact optimal hierarchical clustering among all those represented in the trellis structure.

Theorem 1. (Correctness) *Given a trellis, $\mathbb{T}$, dataset, X, and objective-specific admissible heuristic function, $\mathbb{H}$, Algorithm 1 yields $H^\star = \operatorname{argmin}_{H \in \mathbb{H}(\mathbb{T})} \phi(H)$ where $\mathbb{H}(\mathbb{T})$ is the set of all trees represented by the trellis.*

See Appendix §A.1 for proof.

Corollary 2. (Optimal Clustering) *Given a dataset, X, let the trellis, $\mathbb{T} = \mathbb{P}(X)$, be the powerset. Algorithm 1 yields the (global) optimal hierarchical clustering for X.*

See Appendix §A.2 for proof.

Next, we consider the space and time complexities of Algorithm 1. We observe that the algorithm scales with the size of the trellis structure, and requires at most exponential space and time, rather than scaling with the number of trees, $(2n-3)!!$, which is super-exponential.

Theorem 3. (Space Complexity) *For trellis, $\mathbb{T}$ and dataset, X, Algorithm 1 finds the lowest cost hierarchical clustering of X present in $\mathbb{T}$ in space $O(|\mathbb{T}|^2)$, which is at most $O(3^{|X|})$ when $\mathbb{T} = \mathbb{P}(X)$.*

See Appendix §A.3 for proof.

8

Algorithm 1 A* Hierarchical Clustering Search

1: **function** SEARCH($\mathbb{T}$, X, $\mathbb{H}$)
2:     **Input:** A trellis structure $\mathbb{T}$ and dataset X, a heuristic function $\mathbb{H}$
3:     **Output:**   The lowest cost tree represented in the trellis.
4:     **do**
5:         ▷ Get state from trellis (Eq. 2)
6:         $H_\Xi(X) = \{X\} \bigcup_{X_c \in X[\Xi]} H_\Xi(X_c)$
7:         $f_{\text{at root}}, \_, h_{\text{at root}}, \_, \_ \leftarrow X[\Pi].\mathsf{peek}()$
8:         ▷ At goal state?
9:         **if** $h_{\text{at root}} = 0$ **and** $|\mathsf{lvs}(H_\Xi(X))| = |X|$ **then**
10:            **return** $f_{\text{at root}}, H_\Xi(X)$
11:        ▷ Explore new leaves
12:        **for** X_i **in** $\mathsf{lvs}(H_\Xi(X))$ **do**
13:            **for** $X_L,\ X_R$ **in** CHILDREN(X_i) **do**
14:                Define g_{LR} according to Eq. 4
15:                Define h_{LR} according to Eq. 5
16:                $f_{LR} \leftarrow g_{LR} + h_{LR}$
17:                $X_i[\Pi].\mathsf{enqueue}((f_{LR}, g_{LR}, h_{LR}, X_L,\ X_R))$
18:        ▷ Update each node in the tree's min heap
19:        **for** X_i **in** $H_\Xi(X)$ **from leaves to root do**
20:            $\_, \_, \_, X_L,\ X_R \leftarrow X_i[\Pi].\mathsf{pop}()$
21:            Define $g \leftarrow g_{LR}$ according to Eq. 4
22:            $h \leftarrow 0$
23:            **for** X_c **in** $[X_L, X_R]$, **do**
24:                **if** $X_c[\Pi]$ **is defined then**
25:                    $\_, g_c, h_c, \_,\ \_ \leftarrow X_c[\Pi].\mathsf{peek}()$
26:                    $g \leftarrow g + g_c$
27:                    $h \leftarrow h + h_c$
28:                **else**
29:                    $h \leftarrow h + \mathbb{H}(X_c)$
30:            ▷ Update the f value of split X_L, X_R
31:            $X_i[\Pi].\mathsf{enqueue}((g + h, g, h, X_L, X_R))$
32:     **while** True

Theorem 4. Time Complexity *For trellis, $\mathbb{T}$ and dataset, X, Algorithm 1 finds the lowest cost hierarchical clustering of X in time $O(|\{\ \mathrm{CH}(X, \mathbb{T}) \mid X \in \mathbb{T}\}|)$. which is at most $O(3^{|X|})$ when $\mathbb{T} = \mathbb{P}(X)$.*

See Appendix §A.4 for proof.

Finally, we observe that Algorithm 1 is optimally efficient when given a consistent / monotone heuristic.

Theorem 5. Optimal Efficiency *For trellis, $\mathbb{T}$ and dataset, X, Algorithm 1 is optimally efficient if h is a consistent / monotone heuristic.*

See Appendix § A.5 for proof.

4 Trellis Construction

When running A*, the exploration step at node X_i requires the instantiation of a min heap of node X_i's children. As described in §3, we can use Algorithm 1 to find exact solutions if we search the full trellis structure that includes all subsets of the dataset, $\mathbb{P}(X)$. However, we are also interested in approximate methods. Here, we propose three approaches: (1) run A* using a sparse trellis (one with missing nodes and/or edges), (2) extend a sparse trellis by adding nodes, and edges corresponding to child / parent relationships while running A* at exploration time for each node explored during A* search, (3) run A* iteratively, obtaining a solution and then running A* again, extending the trellis further between or during subsequent iterations.

Trellis Initialization Before running A*, an input structure defining the children / parent relationships and nodes in the trellis can be provided. One possibility is to use an existing method to create this structure. For instance, it is possible to initialize all the nodes coming from the full/partial beam size set of hierarchies obtained from running beam search. However, this structure can be updated during run-time using the method described below by adding additional children to a node beyond those present at initialization. This way, A* will include within the search space, hierarchies coming from small perturbations (at every level) of the ones employed to initialize the trellis.

Running A* While Extending The Trellis A sparse trellis (even one consisting solely of the root node) can be extended during the exploration stage of A* search. Given a node, X_i, in a sparse trellis, sample the children to place on X_i's queue using an objective function-based sampling scheme. It is reasonable to randomly sample a relatively large number of candidate children of the node and then either restrict the children to the K best according to the value of the $\psi(\cdot, \cdot)$ function of the cost (Eq. 1) (best K sampling) or sample from the candidate children according to their relative probability, i.e., $\psi(\cdot, \cdot) / \sum \psi(\cdot, \cdot)$ (importance sampling).

Iterative A*-based Trellis Construction We can combine the methods above, by initializing a sparse trellis, extending it during run-time, and then run an iterative algorithm that outputs a series of hierarchical clusterings with monotonically decreasing cost. It uses $\mathbb{T}^{(r)}$ as the initialization, runs A* and outputs the best hierarchical clustering represented by that trellis. In each subsequent round $r + 1$, $\mathbb{T}^{(r)}$ is extended at run-time, adding more nodes and edges, and at the end of the round, outputs the best hierarchical clustering

represented by trellis $\mathbb{T}^{(r+1)}$. This can be repeated until some stopping criteria are reached, or until the full trellis is created.

5   Jet Physics Experiments

Additional Background. Detectors at the Large Hadron Collider (LHC) at CERN measure the energy (and momentum) of particles generated from the collision of two beams of high-energy protons. Typically, the pattern of particle hits will have localized regions. Recall the particles in each region are clustered (i.e., a jet), and this hierarchical clustering is originated by a *showering process* where the initial (unstable) particle (root) goes through successive binary splittings until reaching the final state particles (with an energy below a given threshold) that hit the detector and are represented by the leaves. These leaves are observed while the latent showering process, described by quantum chromodynamics, is not. This showering-process is encoded in sophisticated simulators, and rather than optimizing heuristics (as is traditionally done, e.g., [6, 7, 8, 9]), directly maximizing the likelihood, (1) allows us to compute MAP hierarchical clusterings generated by these simulators, (2) unifies generation and inference, and (3) can improve our understanding of particle physics.

Cost Function. We use a cost function that is the negative log-likelihood of a model for jet physics [29]. 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 Ginkgo, a prototypical model for jet physics [29] that provides a tractable joint likelihood, where for each pair of parent and left (right) child cluster with masses $\sqrt{t_P}$ and $\sqrt{t_L}$ ($\sqrt{t_R}$) respectively, the likelihood function is,

$$\psi(X_L, X_R) = f(t(x_L)|t_P, \lambda) \cdot f(t(x_R)|t_P, \lambda) \tag{6}$$

$$\text{with} \quad f(t|t_P, \lambda) = \frac{1}{1 - e^{-\lambda}} \frac{\lambda}{t_P} e^{-\lambda \frac{t}{t_P}} \tag{7}$$

where the first term in $f(t|t_P, \lambda)$ is a normalization factor associated to the constraint that $t < t_P$, and λ a constant. For the leaves, we need to integrate $f(t|t_P, \lambda)$ from 0 to the threshold constant t_{cut} (see [29] for more details),

$$f(t_{cut}|\lambda, t_P) = \frac{1}{1 - e^{-\lambda}} \left(1 - e^{-\frac{\lambda}{t_P} t_{cut}} \right) \tag{8}$$

Heuristic function. We introduce two heuristic functions to set an upper bound on the log likelihood of Ginkgo hierarchies, described in detail in Appendix § A.7. We provide a brief description as follows. We split the heuristic into internal nodes and leaves. For the leaves, we roughly bound t_P in Equation 8 by the minimum squared mass, among all the nodes with two elements, that is greater than t_{cut} (see [29] for more details). For internal nodes, we wish to find the smallest possible upper bound to Equation 7. We bound t_P in the

exponential by the squared mass of the root node (top of the tree). Next, we look for the biggest lower bound on the squared mass t in Equation 7. We consider a fully unbalanced tree as the topology with the smallest per level values of t and we bound each level in a bottom up approach (singletons are at the bottom). Finally, to get a bound for t_P in the denominator of Equation 7 we want the minimum possible values. Here we consider two heuristics:

- Admissible $\mathsf{h0}(\cdot)$. We take the minimum per level parent squared mass plus the minimum leaf squared mass, as the parent of every internal node has one more element.

- Approximate $\mathsf{h1}(\cdot)$. We take the minimum per level parent squared mass plus 2 times the minimum squared mass among all the nodes with two elements.

In principle, $\mathsf{h1}(\cdot)$ is approximate, but we studied its effectiveness by checking that the cost was always below the exact trellis MAP tree on a dataset of 5000 Ginkgo jets with up to 9 elements, as well as the fact that exact A* with $\mathsf{h1}(\cdot)$ agrees with the exact trellis within 6 significant figures (see Figure 5). In any case, if for some cases $\mathsf{h1}(\cdot)$ is inadmissible, the only downside is that we could be missing a lower cost solution for the MAP tree. As a result, given that $\mathsf{h1}(\cdot)$ is considerably faster than $\mathsf{h0}(\cdot)$, below we show results for A* implemented with $\mathsf{h1}(\cdot)$.

Data and Methods. We compare our algorithm with different benchmarks. We start with greedy and beam search implementations as baselines, where we cluster Ginkgo jet constituents (leaves of a binary tree) based on the joint likelihood to get the maximum likelihood estimate (MLE) for the hierarchy that represents the latent structure of a jet. 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 1000 for datasets with more than 40 elements or $\frac{N(N-1)}{2}$ for the others, with N the number of elements to cluster. Also, we discarded partial clusterings with identical likelihood values, to account for the different orderings of a hierarchy (see [30] for more details), which significantly improved the performance of beam search. As a stronger baseline, we implement the Monte-Carlo Tree Search (MCTS) algorithm for hierarchical clusterings introduced in [31]. We choose the best performing case, that corresponds to a neural policy trained for 60000 steps, initializing the search tree at each step by running beam search with beam size 100. Also, the maximum number of MCTS evaluation roll-outs is 200. Finally, for exact solutions, we compare with the exact hierarchical cluster trellis algorithm introduced in [21].

Approximate A*. We design the approximate A*-based method following the details described in Section 4. We first initialize the sparse trellis the full set of beam search hierarchies (§4). Next, we use a top K-based sampling (§4) and the iterative construction procedure (§4) to extend the search space.
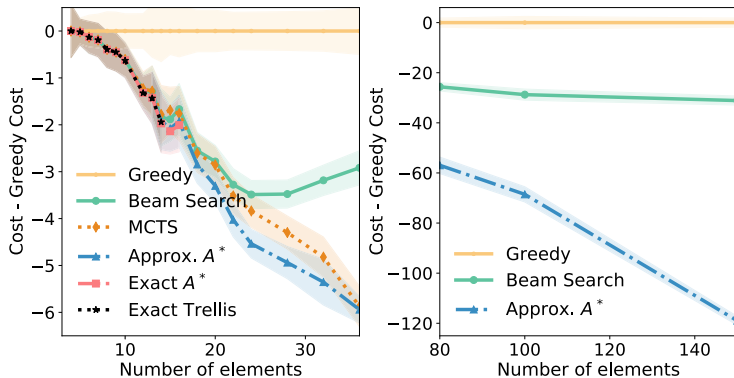
Figure 5: **Jet Physics Results.** Cost (Neg. log. likelihood) for the MAP hierarchy of each algorithm on Ginkgo datasets minus the cost for greedy (lower values are better solutions). We see that both exact and approx. A* greatly improve over beam search and greedy. Though MCTS provides a strong baseline for small datasets where it is feasible to implement it (left), A* still shows an improvement over it.

Figure 5 shows a comparison of the MAP values of the proposed exact and approximate algorithms using A* with benchmark algorithms (greedy, beam search, MCTS, and exact trellis), on Ginkgo jets. For the A*-based method we show both the exact algorithms and approximate solutions, both implemented with the heuristic denoted as $\mathsf{h1}(\cdot)$ in Appendix § A.7. We want to emphasize that approximate versions of A* allow the algorithm to handle much larger datasets while significantly improving over beam search and greedy baselines. MCTS provides a strong baseline for small datasets, where it is feasible to implement it. However, A* shows an improvement over MCTS while also being feasible for much larger datasets. Next in Figure 6 we show the empirical running times. We want to point out that the A*-based method becomes faster than the exact trellis one for data with 12 or more elements, which makes A* feasible for larger datasets. We can see that approx. A* follows the exact A* cost very closely starting at 12 elements until 15, while having a lower running time. Though approx. A* and MCTS have a running time of the same order of magnitude (only evaluation time for MCTS) for more than 20 leaves, A* has a controlled running time while MCTS evaluation time grows exponentially.

The exact trellis time complexity is $\mathcal{O}(3^N)$ which is super-exponentially more efficient than brute force methods that consider every possible hierarchy and grow at the rate of $(2N-3)!!$. Thus, in Figure 7 we show the number of hierarchies explored by A* divided by 3^N, and we can see that exact and approximate A* are orders of magnitude more efficient than the trellis, which to the best of our knowledge was the most efficient exact algorithm at present. Also, we note as a point of reference that for about 100 leaves, the approx. A* results of Figure 7 can improve over benchmarks by only exploring $\mathcal{O}(10^{-180})$ of the possible hierarchies.
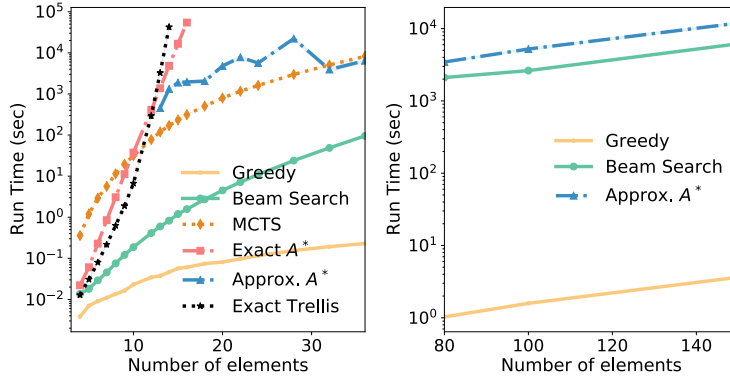
Figure 6: **Running Times.** Empirical running time on an Intel Xeon Platinum 8268 24C 2.9GHz Processor. The A*-based method becomes faster than the exact trellis one for data with 12 or more elements. Approx. A* is much faster than exact algorithms while still significantly improving over the baselines. This figure only shows evaluation running times for MCTS, after having trained the model for 7 days with the same processor. Evaluation MCTS times grow exponentially while approx. A* has a controlled run time.
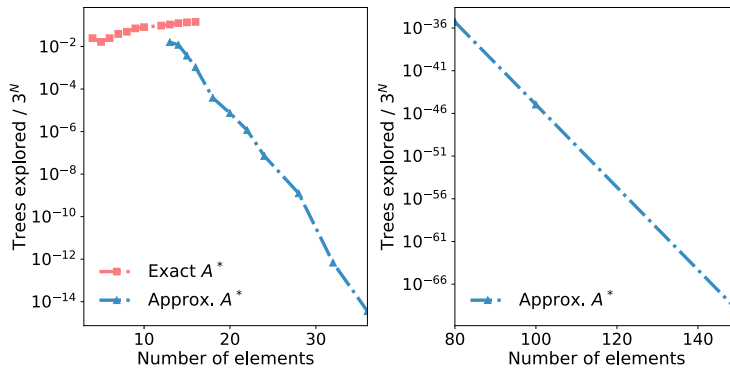

Figure 7: **Search Space Exploration.** Number of hierarchies explored divided by the exact trellis time complexity, i.e 3^N. We see that the A*-based method is considerably more efficient than the trellis, which, in turn, is super-exponentially more efficient than an explicit enumeration of clusterings.
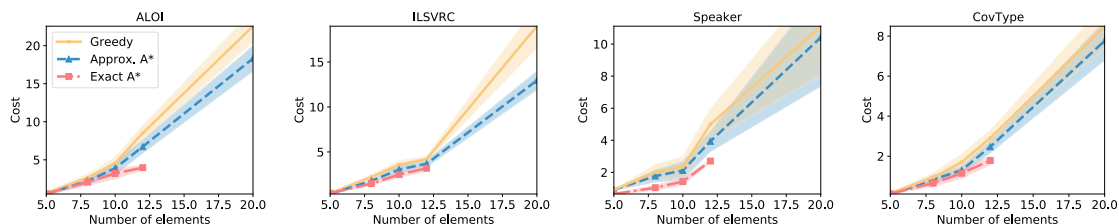
14

Figure 8: **Hierarchical Correlation Clustering on Benchmark Data**. On randomly sampled subsets small enough for us to run the exact A* approach, we report the hierarchical correlation clustering cost (lower is better) for clustering benchmarks. We observe that the approximate A* method is able to achieve lower cost clusterings compared to greedy.
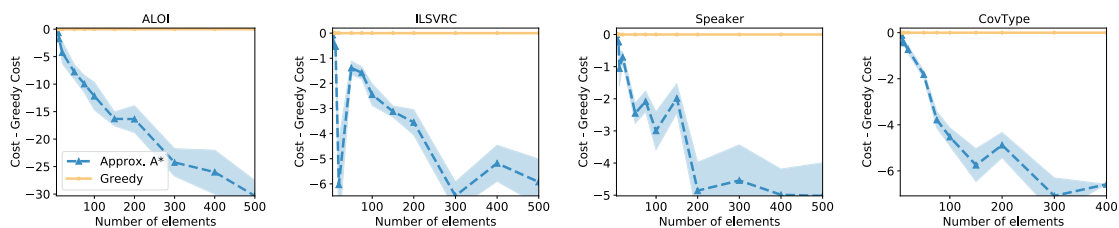

Figure 9: **Improvement over Greedy on Larger Benchmark Datasets** . We report the improvement (cost, lower is better) of the approximate A* method over the greedy baseline.

15

6   Experiments on Clustering Benchmarks

In this section, we analyze the performance of A* on several commonly used hierarchical clustering benchmark datasets [26, 13]: **Speaker**, i-vector speaker recordings, ground truth clusters refer each unique speaker [32]; **CovType**, a dataset of forest cover types; **ALOI**, color histogram features of 3D objects, ground truth clusters refer to each object type [33]; **ILSVRC**, image features from InceptionV3 [34] of the ImageNet ILSVRC 2012 dataset [35].

In this experiment, we use cosine similarity as the measure of the pairwise similarity between points, as it is known to be meaningful for each of the datasets [13], and hierarchical correlation clustering as the cost function:

Objective 1. (Hierarchical Correlation Clustering) *Following [21], we consider a hierarchical version of well-known flat clustering objective, correlation clustering [36]. In this case, we define the cost of a pair of sibling nodes in the tree to be the sum of the positive edges crossing the cut, minus the sum of the negative edges not crossing the cut:*

$$\psi(X_i, X_j) = \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \mathbb{I}[w_{ij} > 0] + \sum_{\substack{x_i, x_j \in X_i \times X_i, \\ i < j}} |w_{ij}| \mathbb{I}[w_{ij} < 0] + \sum_{\substack{x_i, x_j \in X_j \times X_j, \\ i < j}} |w_{ij}| \mathbb{I}[w_{ij} < 0] \qquad (9)$$

where w_{ij} is the affinity between x_i and x_j.

To build the weighted graphs needed as input to correlation clustering, we subtract the mean similarity from each of the pairwise similarities.

We can provide a lower bound for this objective by using the sum of the of the positive edges contained in the dataset,

$$h_{\mathsf{cc}}(X) = \sum_{\substack{x_i, x_j \in X \times X, \\ i < j}} w_{ij} \mathbb{I}[w_{ij} > 0]. \qquad (10)$$

Proposition 1. (Admissible Heuristic for Hierarchical Correlation Clustering) *Equation 10 is an admisslbe heuristic for Hierarchical Correlation Clustering cost, that is $h_{\mathsf{cc}}(X) \leq \phi_{HCC}(X)$.*

See Appendix §A.6 for proof.

Approximate A*. In this experiment, we apply the techniques described in Section 4 to design an approximation algorithm. We first initialize the sparse trellis using a greedy approach (§4). We then use a top K-based sampling (§4) and iterative construction approach (§4) to extend the search space beyond greedy initialization and use heuristic h_{cc}.

Figure 8, shows the cost (lower is better) of the exact solution found via A*, the aforementioned approximate A*, and the greedy solution on small datasets. We sample each of the small datasets from their corresponding original dataset using stratified sampling, where the strata correspond to the ground truth class labels. We report the mean and

16

standard deviation / sqrt(num runs) computed over 5 random samples of each dataset size. We observe that the approximate method is able to provide hierarchical clusterings that have lower cost than the greedy approach, nearing the exact MAP values found by the optimal approach. In Figure 9, we show results on data sets with a larger number of samples (sampled from the original datasets in the same manner as before), which are too large to run the exact algorithm. We plot the reduction (mean, standard deviation / sqrt(num runs) over 5 runs per dataset size) in cost of the clusterings found by the approximate A* algorithm vs. the greedy approach.

7   Related Work

An A*-based approach is used in [24] to find MAP solutions to Dirichlet Process Mixture Models. This work focuses on flat clustering rather than hierarchical, and it does not include a detailed analysis of the time and space complexity, as is done in this paper. In contrast to our deterministic search-based approach, sampling-based methods, such as [37, 38, 39, 30], use MCMC, Metropolis Hastings or Sequential Monte Carlo methods to draw samples from a posterior distribution over trees. Other methods use incremental constructions of tree structures [26, 13, 40], similar to our approach, however they perform tree re-arrangements to overcome greedy decisions made in the incremental setting. These operations are made in a relatively local fashion without considering a global cost function for tree structures as is done in our paper. Linear, quadratic, and semi-definite programming have also been used to solve hierarchical clustering problems [41, 42], as have, branch-and-bound based methods [43]. These methods often have approximation guarantees. It would be interesting to discover how these methods could be used to explore nodes in a trellis structure (§4) in future work. In contrast to our search-based method that considers discrete tree structures, recent work has considered continuous representations of tree structure to support gradient-based optimization [44, 45]. Finally, we note that much recent work, [46, 47, 48, 49, 50, 42], has been devoted to Dasgupta's cost for hierarchical clustering [28]. We direct the interested reader to the Appendix §A.8, for a description of the cost and an admissible heuristic that enables the use of A* to find optimal Dasgupta cost hierarchical clusterings.

8   Conclusion

In this paper, we present a way to run A* to provide exact and approximate hierarchical clusterings. We describe a data structure with a nested representation of the A* search space to provide a time and space efficient approach to search. We demonstrate the effectiveness of the approach experimentally and prove theoretical results about its optimality and efficiency. In future work, we hope to extend the same kinds of algorithms to flat clustering costs as well as to probabilistic models in Bayesian non-parametrics.

Acknowledgements

References

[1] T. Sørlie, C. M. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. B. Eisen, M. Van De Rijn, S. S. Jeffrey, *et al.*, "Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications," *Proceedings of the National Academy of Sciences*, vol. 98, no. 19, pp. 10869–10874, 2001.

[2] Y. Zhang, A. Ahmed, V. Josifovski, and A. Smola, "Taxonomy discovery for personalized recommendation," in *Proceedings of the 7th ACM international conference on Web search and data mining*, ACM, 2014.

[3] S. Green, N. Andrews, M. R. Gormley, M. Dredze, and C. D. Manning, "Entity clustering across languages," in *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2012.

[4] M. Levin, S. Krawczyk, S. Bethard, and D. Jurafsky, "Citation-based bootstrapping for large-scale author disambiguation," *Journal of the American Society for Information Science and Technology*, 2012.

[5] S. Vashishth, P. Jain, and P. Talukdar, "Cesi: Canonicalizing open knowledge bases using embeddings and side information," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 1317–1327, International World Wide Web Conferences Steering Committee, 2018.

[6] M. Cacciari, G. P. Salam, and G. Soyez, "The anti-k_t jet clustering algorithm," *JHEP*, vol. 04, p. 63, 2008.

[7] S. Catani, Y. L. Dokshitzer, M. H. Seymour, and B. R. Webber, "Longitudinally invariant k_t clustering algorithms for hadron hadron collisions," *Nucl. Phys. B*, vol. 406, pp. 187–224, 1993.

[8] Y. L. Dokshitzer, G. D. Leder, S. Moretti, and B. R. Webber, "Better jet clustering algorithms," *JHEP*, vol. 08, p. 1, 1997.

[9] S. D. Ellis and D. E. Soper, "Successive combination jet algorithm for hadron collisions," *Phys. Rev. D*, vol. 48, pp. 3160–3166, 1993.

[10] M. Bateni, S. Behnezhad, M. Derakhshan, M. Hajiaghayi, R. Kiveris, S. Lattanzi, and V. Mirrokni, "Affinity clustering: Hierarchical clustering at scale," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[11] H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler, "Bico: Birch meets coresets for k-means clustering," in *European Symposium on Algorithms*, Springer, 2013.

[12] A. Garg, A. Mangla, N. Gupta, and V. Bhatnagar, "Pbirch: A scalable parallel clustering algorithm for incremental data," in *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, pp. 315–316, IEEE, 2006.

[13] N. Monath, A. Kobren, A. Krishnamurthy, M. R. Glass, and A. McCallum, "Scalable hierarchical clustering with tree grafting," in *Knowledge Discovery & Data Mining (KDD)*, 2019.

[14] S. Naumov, G. Yaroslavtsev, and D. Avdiukhin, "Objective-based hierarchical clustering of deep embedding vectors," *arXiv preprint arXiv:2012.08466*, 2020.

[15] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: A new data clustering algorithm and its applications," *Data Mining and Knowledge Discovery*, 1997.

[16] A. Dubey, Q. Ho, S. Williamson, and E. P. Xing, "Dependent nonparametric trees for dynamic hierarchical clustering," *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[17] Z. Hu, H. Qirong, A. Dubey, and E. Xing, "Large-scale distributed dependent nonparametric trees," *International Conference on Machine Learning (ICML)*, 2015.

[18] N. Monath, A. Dubey, G. Guruganesh, M. Zaheer, A. Ahmed, A. McCallum, G. Mergen, M. Najork, M. Terzihan, B. Tjanaka, *et al.*, "Scalable bottom-up hierarchical clustering," *arXiv preprint arXiv:2010.11821*, 2020.

[19] K. A. Dubey, M. Zhang, E. Xing, and S. Williamson, "Distributed, partially collapsed mcmc for bayesian nonparametrics," in *International Conference on Artificial Intelligence and Statistics*, 2020.

[20] N. Monath, M. Zaheer, K. A. Dubey, A. Ahmed, and A. McCallum, "Dag-structured clustering by nearest neighbors," in *International Conference on Artificial Intelligence and Statistics*, 2021.

[21] C. S. Greenberg, S. Macaluso, N. Monath, J.-A. Lee, P. Flaherty, K. Cranmer, A. Mc-Gregor, and A. McCallum, "Data Structures \& Algorithms for Exact Inference in Hierarchical Clustering," 2 2020.

[22] M. Cacciari, G. P. Salam, and G. Soyez, "FastJet User Manual," *Eur. Phys. J. C*, vol. 72, p. 1896, 2012.

[23] A. Butter *et al.*, "The Machine Learning Landscape of Top Taggers," *SciPost Phys.*, vol. 7, p. 014, 2019.

[24] H. Daume III, "Fast search for dirichlet process mixture models," in *Artificial Intelligence and Statistics*, pp. 83–90, 2007.

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

[26] A. Kobren, N. Monath, A. Krishnamurthy, and A. McCallum, "A hierarchical algorithm for extreme clustering," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 255–264, 2017.

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

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

[29] K. Cranmer, S. Macaluso, and D. Pappadopulo, "Toy Generative Model for Jets," 2019.

[30] L. Boyles and M. Welling, "The time-marginalized coalescent prior for hierarchical clustering," in *Advances in Neural Information Processing Systems*, pp. 2969–2977, 2012.

[31] J. Brehmer, S. Macaluso, D. Pappadopulo, and K. Cranmer, "Hierarchical clustering in particle physics through reinforcement learning," in *34th Conference on Neural Information Processing Systems*, 11 2020.

[32] C. S. Greenberg, D. Bansé, G. R. Doddington, D. Garcia-Romero, J. J. Godfrey, T. Kinnunen, A. F. Martin, A. McCree, M. Przybocki, and D. A. Reynolds, "The nist 2014 speaker recognition i-vector machine learning challenge," in *Odyssey: The Speaker and Language Recognition Workshop*.

[33] J.-M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders, "The amsterdam library of object images," in *International Journal of Computer Vision (IJCV)*, 2005.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," in *International Journal of Computer Vision (IJCV)*, 2015.

[36] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," *Machine learning*, vol. 56, no. 1-3, pp. 89–113, 2004.

[37] A. Bouchard-Côté, S. Sankararaman, and M. I. Jordan, "Phylogenetic inference via sequential monte carlo," *Systematic biology*, vol. 61, no. 4, pp. 579–593, 2012.

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

[39] Z. Ghahramani, M. I. Jordan, and R. P. Adams, "Tree-structured stick breaking for hierarchical data," in *Advances in neural information processing systems*, pp. 19–27, 2010.

[40] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," *ACM sigmod record*, vol. 25, no. 2, pp. 103–114, 1996.

[41] S. Gilpin, S. Nijssen, and I. Davidson, "Formalizing hierarchical clustering as integer linear programming," *AAAI*, 2013.

[42] A. Roy and S. Pokutta, "Hierarchical clustering via spreading metrics," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 3077–3111, 2017.

[43] C. Cotta and P. Moscato, "A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny," *Biosystems*, vol. 72, no. 1-2, pp. 75–97, 2003.

[44] N. Monath, M. Zaheer, D. Silva, A. McCallum, and A. Ahmed, "Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space," *KDD*, 2019.

[45] I. Chami, A. Gu, V. Chatziafratis, and C. Ré, "From trees to continuous embeddings and back: Hyperbolic hierarchical clustering," *NeurIPS*, 2020.

[46] M. Charikar and V. Chatziafratis, "Approximate hierarchical clustering via sparsest cut and spreading metrics," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 841–854, SIAM, 2017.

[47] M. Charikar, V. Chatziafratis, and R. Niazadeh, "Hierarchical clustering better than average-linkage," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2291–2304, SIAM, 2019.

[48] V. Chatziafratis, R. Niazadeh, and M. Charikar, "Hierarchical clustering with structural constraints," *arXiv preprint arXiv:1805.09476*, 2018.

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

[50] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu, "Hierarchical clustering: Objective functions and algorithms," *Journal of the ACM (JACM)*, vol. 66, no. 4, pp. 1–42, 2019.

A Appendix

A.1 Theorem 1 (Correctness of A* MAP) Proof

Proof. Assume toward contradiction that Algorithm 1 does not return the optimal hierarchical clustering, $H^\star$. It is clear from the conditional in line 9, $|\mathsf{lvs}(H_\Xi(X))| = |X|$ and the fact that if X_L, X_R are children of X then $X_L \bigcup X_R = X$ and $X_L \bigcap X_R = \varnothing$, that Algorithm 1 returns a hierarchical clustering, $H_\Xi(X) \in \mathbb{H}(\mathbb{T})$, so the assumption is that $H_\Xi(X)$ is not optimal, i.e., $\phi(H^\star) < \phi(H_\Xi(X))$. Consider the set, S, of data sets present in both $H^\star$ and $H_\Xi(X)$ that branch to different children, i.e., $S = \{X_i | X_{L^\star}, X_{R^\star} \in H^\star \wedge X_L, X_R \in H_\Xi(X) \wedge X_{L^\star} \bigcup X_{R^\star} = X_L \bigcup X_R = X_i \wedge X_{L^\star} \neq X_L \neq X_R \neq X_{R^\star}\}$. The set S must be non empty, otherwise $H_\Xi(X) = H^\star$, a contradiction. Now consider the set, T, of ancestors of S, i.e., $T = \{X_i | X_i \in S \wedge \nexists X_j \in S \text{ s.t. } X_i \neq X_j \wedge X_i \bigcap X_j = X_i\}$. Since S is non-empty, the set T must also be non empty. Now select a dataset, $X_i \in T$, such that the sub-hierarchy rooted at X_i in $H^\star$, $H^\star(X_i)$, has lower energy than the sub-hierarchy rooted at X_i in $H_\Xi(X)$, $H_\Xi(X_i)$, i.e., $\phi(H^\star(X_i)) < \phi(H_\Xi(X_i))$. There must exist at lease one such X_i, otherwise $\phi(H_\Xi(X)) <= \phi(H^\star)$, a contradiction.

Now consider the trellis vertex corresponding to data set X_i, $\mathbb{V}_i$, with min heap $X_i[\Pi]$, and the tuple at the top of $X_i[\Pi]$, $(f_i, g_i, h_i, X_L, X_R)$. Note that because $h = 0$ in line 9, that $h_i = 0$, therefore the f_i is equal to the energy of the sub-hierarchy of $H_\Xi(X)$ rooted at X_i, $\phi(H_\Xi(X_i))$. Note also that X_i's children in $H^\star$, $X_{L^\star}, X_{R^\star}$, must be present in $X_i[\Pi]$, along with corresponding $f^\star, g^\star, h^\star$ values, otherwise $H^\star$ is not represented in the trellis, a contradiction. Since $X_{L^\star} \neq X_L \neq X_R \neq X_{R^\star}$, the tuple $(f^\star, g^\star, h^\star, X_{L^\star}, X_{R^\star})$ is not at the top of $X_i[\Pi]$. If $h^\star = 0$, then $f^\star = \phi(H^\star(X_i))$ and $f <= f^\star$, which implies $\phi(H_\Xi(X_i)) <= \phi(H^\star(X_i))$, a contradiction. If $h^\star \neq 0$, then $f^\star <= \phi(H^\star(X_i))$, otherwise $\mathbb{H}$ is not an admissible heuristic, a contradiction. This gives us $\phi(H_\Xi(X_i)) = f < f^\star <= \phi(H^\star(X_i))$, a contradiction. $\square$

A.2 Corollary 2 (Optimal Clustering) Proof

Proof. Theorem 1 states that Algorithm 1 returns the optimal hierarchical clustering, thus proving that all hierarchical clusterings are represented in a full trellis would prove the corollary.

Assume toward contradiction that there is a hierarchical clustering, $H^\star$, that is not present in a full trellis, $\mathbb{T}$. This implies either that (1) there is a data set $X_i \in H^\star$ that is not in the trellis, i.e., $X_i \notin \mathbb{T}$, or that (2) there exists data sets $X_i, X_j, X_k \in H^\star$ s.t. $X_i = X_j \bigcup X_k \wedge X_j \bigcap X_k = \varnothing$ and $X_i, X_j, X_k \in \mathbb{T}$, but $X_j, X_k \notin \mathbb{C}(X_i)$. Regarding (1), $X_i \in H^\star$ s.t. $X_i \notin \mathbb{T}$ implies that $X \notin \mathbb{P}(X)$, a contradiction. Regarding (2), since each node in a full trellis has all possible children, i.e., $\forall \mathbb{V}_i \in \mathbb{T}, \mathbb{C}(\mathbb{V}_i) = \mathbb{P}(X_i)$, this implies that $X_j, X_k \notin \mathbb{P}(X_i)$, a contradiction.

$\square$

A.3 Theorem 3 (Space Complexity of A* MAP) Proof

Proof. Each vertex, $\mathbb{V}_i$, in a trellis, $\mathbb{T}$, stores a dataset, X_i, a min heap, $X_i[\Pi]$, and a pointer to the children associated with the best split, $X[\Xi]$. Since X_i and each pointer in $X[\Xi]$ can be represented as integers, they are stored in $O(1)$ space. The min heap, $X_i[\Pi]$, stores a five tuple, $(f_{LR}, g_{LR}, h_{LR}, X_L, X_R)$ for each of X_i's child pairs, (X_L, X_R). Each five tuple can be stored in $O(1)$ space, so the min heap is stored in $O(|\mathbb{C}(X_i)|)$ space, where $\mathbb{C}(X_i)$ is X_i's children. Thus each vertex, $\mathbb{V}_i$, takes $O(|\mathbb{C}(X_i)|)$ space to store. In a full trellis, $|\mathbb{C}(X_i)| = 2^{|X_i|-1} - 1$, making the total space complexity $\sum_{\mathbb{V} \in \mathbb{T}} 2^{|X_i|} = O(3^{|X|})$. In a sparse trellis, the largest number of parent/child relationships occurs when the trellis is structured such that the root, $\mathbb{V}_1$, is a parent of all the remaining $|\mathbb{T}| - 1$ vertices in the trellis, the eldest child of the root, $\mathbb{V}_2$, is a parent of all the remaining $|\mathbb{T}| - 2$ vertices in the trellis, and so on, thus the space complexity of a sparse trellis is $\sum_{\mathbb{V}_i \in \mathbb{T}} O(|\mathbb{C}(X_i)|) <= \sum_{i=1...|\mathbb{T}|} (|\mathbb{T}| - i) = O(|\mathbb{T}|^2)$. $\qquad\square$

A.4 Theorem 4 (Time Complexity of A* MAP) Proof

Proof. During each iteration of the loop beginning at line 4, the algorithm descends from the root down to the leaves of the partial hierarchical clustering with minimum energy, $H_\Xi(X)$, such that the internal nodes have all been explored and the leaf nodes have not. Upon arriving at the leaves of the partial hierarchical clustering (line 11), each leaf node is explored, creating and populating a min heap stored at the node. Finally in line 19, every node in $H_\Xi(X)$ updates its min heap.

Note that when running Algorithm 1, each node in the trellis is explored at most once (a given node, $\mathbb{V}_i$, is explored when $\mathbb{V}_i \in lvs(X[\Xi])$ at line 12). If every node in a trellis is explored, $H_\Xi(X) = \arg\min_{H \in \mathbb{H}(X)}$, at which point Algorithm 1 computes $H_\Xi(X)$ at line 6 in $O(\log n)$ time and halts at line 10. Therefore, the time it takes Algorithm 1 to explore every node in trellis $\mathbb{T}$ gives an upper bound on the time it takes Algorithm 1 to find the tree with minimum energy hierarchical clustering encoded by $\mathbb{T}$.

The time it takes Algorithm 1 to explore every $\mathbb{V}_i \in \mathbb{T}$ is the sum of (1) the time $\mathbb{V}_i$ contributes to computation of $H_\Xi(X)$ at line 6 when $\mathbb{V}_i \in lvs(X[\Xi])$, (2) the time it takes to populate the min heap for each $\mathbb{V}_i$ at line 13, and (3) the time $\mathbb{V}_i$ contributes to updating the min heap of every node in $H_\Xi(X)$ at line 19 when $\mathbb{V}_i \in lvs(X[\Xi])$.

(1) The time it takes for Algorithm 1 to compute $H_\Xi(X)$ in a given iteration is the sum of the length of the root to leaf paths in $lvs(X[\Xi])$, since reading $X[\Xi]$ takes $O(1)$ time. Therefore the amount of time $\mathbb{V}_i$ contributes to computation of $H_\Xi(X)$ is the length of the path in $H_\Xi(X)$ from the root to $\mathbb{V}_i$. In a full trellis, the maximum possible path length from the root to $\mathbb{V}_i$ is $|X| - |X_i|$. Thus, the total amount of time spent computing $H_\Xi(X)$ at line 6 in a full trellis when every vertex is explored is $\sum_{k=1...n} \binom{n}{k}(n - k) = \frac{1}{2}\frac{(2^n - 2)}{n} = O(2^n)$. In a sparse trellis, the path lengths are maximized when the root, $\mathbb{V}_1$, is a parent of all the remaining $|\mathbb{T}| - 1$ vertices in the trellis, the eldest child of the root, $\mathbb{V}_2$, is a parent of all the

24

remaining $|\mathbb{T}| - 2$ vertices in the trellis, and so on. Thus the sum of the total path lengths is bounded from above by $\sum_{i=1...|\mathbb{T}|} (|\mathbb{T}| - i) = O(|\mathbb{T}|^2)$, and the total amount of time spent computing $H_\Xi(X)$ at line 6 in a sparse trellis when every vertex is explored is $O(|\mathbb{T}|^2)$.

(2) The time it takes for Algorithm 1 to populate the min heap for $\mathbb{V}_i$ is the amount of time it takes to compute f_{LR}, g_{LR}, h_{LR} (lines 14-16) and to enqueue $(f_{LR}, g_{LR}, h_{LR}, X_L, X_R)$ onto the min heap (line 17) for all children, X_L, X_R.

It is possible to compute g_{LR} in $O(1)$ time, since the first term in equation 4 is a value look up in the model, and second and third terms are the g_{LR} values at X_L and X_R, respectively, and are memoized at those nodes. It is possible to compute f_{LR} in $O(1)$ time, since it is just the sum of two numbers, g_{LR} and h_{LR}. The time it takes to compute h_{LR} depends on the objective-specific heuristic being used. Note that h_{LR} is the sum of a function of a single X_i in Equation 5, i.e.,

$$h_{LR} = \sum_{X_\ell \in \mathsf{lvs}(H_\Xi(X_L \cup X_R))} \mathbb{H}(X_\ell)$$
$$= \sum_{X_\ell \in \mathsf{lvs}(H_\Xi(X_L))} \mathbb{H}(X_\ell) + \sum_{X_\ell \in \mathsf{lvs}(H_\Xi(X_R))} \mathbb{H}(X_\ell)$$

so we compute $\mathbb{H}(X_i)$ once per node, memoize it, and compute h_{LR} in $O(1)$ time given $\mathbb{H}(X_L)$ and $\mathbb{H}(X_R)$ by summing the two values.

The time complexity of creating a heap of all tuples is $O(|\mathbb{C}(|\mathbb{V}_i|)$, where $|\mathbb{C}(\mathbb{V}_i)|$ is the number of children $\mathbb{V}_i$ has in trellis $\mathbb{T}$.

Thus it takes $O(|\mathbb{C}(\mathbb{V}_i)|) + O(|\mathbb{H}(X_i)|)$ time to create the min heap for $\mathbb{V}_i$, where $O(|\mathbb{H}(X_i)|)$ is the time complexity for computing $\mathbb{H}(X_i)$.

Therefore, creating the min heaps for every node in a full trellis takes $\sum_{k=1...n} \binom{n}{k} O(2^k) = O(3^n)$ time, when $O(\mathbb{H}(X_i)) = O(2^{|X_i|})$. In a sparse trellis, $\sum_{\mathbb{V}_i \in \mathbb{T}} O(|\mathbb{C}(\mathbb{V}_i)|) + O(|\mathbb{H}(X_i)|) = \sum_{\mathbb{V}_i \in \mathbb{T}} O(|\mathbb{C}(\mathbb{V}_i)|) + \sum_{\mathbb{V}_i \in \mathbb{T}} O(|\mathbb{H}(X_i)|) = \sum_{i=1...|\mathbb{T}|} (|\mathbb{T}| - i) = O(|\mathbb{T}|^2)$, when each trellis vertex, $\mathbb{V}_i$, has the maximum possible number of children and $O(\mathbb{H}(X_i)) = O(|X_i|)$.

(3) When exploring node $\mathbb{V}_i$, Algorithm 1 pops and enqueues an entry from the min heap of every node on the path from $\mathbb{V}_i$ to the root in $H_\Xi(X)$, which takes $O(log(|\mathbb{C}(\mathbb{V}_k)|)$ for each ancestor $\mathbb{V}_k$ in the path. In a full trellis, the maximum possible path length from the root to $\mathbb{V}_i$ is $|X| - |X_i|$. Thus, in the worst case it takes Algorithm 1 $\sum_{j=||X_i|...n} O(log(2^j))$ time to update the min heaps when $\mathbb{V}_i$ is explored. Therefore, the total amount of time spent updating the min heaps in $H_\Xi(X)$ at line 19 in a full trellis when every vertex is explored is $\sum_{k=1...n} \binom{n}{k} \sum_{j=k...n} log(2^k) = O(n^2 2^n)$. In a sparse trellis, the path lengths are maximized when the root, $\mathbb{V}_1$, is a parent of all the remaining $|\mathbb{T}| - 1$ vertices in the trellis, the eldest child of the root, $\mathbb{V}_2$, is a parent of all the remaining $|\mathbb{T}| - 2$ vertices in the trellis, and so on. Thus the sum of the total path lengths is bounded from above by $\sum_{i=1...|\mathbb{T}|} \sum_{j=i...|\mathbb{T}|} log(j) = O(log(|\mathbb{T}|^3))$

Therefore, the time complexity of running Algorithm 1 on a full trellis is $O(3^n)$, and the time complexity of running Algorithm 1 on a full trellis is $O(|\mathbb{T}|^2)$.

□

A.5   Theorem 5 (Optimal Efficiency of A* MAP) Proof

Proof. We define a mapping between paths in the trellis representing the state space and the standard space of tree structures via $H_\Xi(X)$ (Eq. 2). This mapping is bijective. We have describe a method to find the neighboring states of $H_\Xi(X)$ in the standard space of trees using the trellis in Algorithm 1 ($\triangleright$Explore new leaves). Given the additive nature of the cost functions in the family of costs (Eq. 2), we can maintain the splits/merges in the aforementioned nested min heap in the trellis structure and have parent nodes' f values be computed from their child's min heaps. The result is that Algorithm 1 exactly follows A*'s best-first according to f with an admissible heuristic in search over the entire space of tree structures. Therefore the optimal efficiency of Algorithm 1 follows as a result of the optimal efficiency of the A* algorithm. □

A.6   Proposition 1 (Admissibility of Hierarchical Correlation Clustering Cost Heuristic) Proof

Proof. We wish to prove that $\mathbb{H}_{hcc1}(X) <= \arg\min_{H \in \mathbb{H}(X)} \phi_{hcc}(H)$. Note that in every hierarchical clustering, every element is eventually separated (at the leaves), thus every edge eventually crosses a cut in every tree, thus $\forall H \in \mathbb{H}(X)$,

$$\sum_{X_L, X_R \in \mathsf{sibs}(H)} \sum_{x_i, x_j \in X_L \times X_R} w_{ij} \mathbb{I}[w_{ij} > 0] = \sum_{x_i, x_j \in X} w_{ij} \mathbb{I}[w_{ij} > 0] \tag{11}$$

Since

$$\forall X, \sum_{\substack{x_i, x_j \in X \times X, \\ i < j}} |w_{ij}| \mathbb{I}[w_{ij} < 0] >= 0$$

We have

$$\phi(H) = \sum_{X_L, X_R \in \mathsf{sibs}(H)} \psi(X_L, X_R)$$

$$= \sum_{X_L, X_R \in \mathsf{sibs}(H)} \left(\sum_{x_i, x_j \in X_L \times X_R} w_{ij} \mathbb{I}[w_{ij} > 0] + \sum_{\substack{x_i, x_j \in X_L \times X_L, \\ i < j}} |w_{ij}| \mathbb{I}[w_{ij} < 0] + \sum_{\substack{x_i, x_j \in X_R \times X_R, \\ i < j}} |w_{ij}| \mathbb{I}[w_{ij} < 0] \right)$$

$$>= \sum_{X_L, X_R \in \mathsf{sibs}(H)} \sum_{x_i, x_j \in X_L \times X_R} w_{ij} \mathbb{I}[w_{ij} > 0]$$

$$= \sum_{x_i, x_j \in X} w_{ij} \mathbb{I}[w_{ij} > 0] = \mathbb{H}_{hcc1}(X)$$

□

A.7 Heuristic Functions for Ginkgo Jets

We want to find heuristic functions to set an upper bound on the likelihood of Ginkgo hierarchies (for more details about Ginkgo see [29]). We split the heuristic into internal nodes and leaves.

A.7.1 Internal nodes

We want to find the smallest possible upper bound to

$$f(t|\lambda, t_{\mathrm{P}}) = \frac{1}{1 - e^{-\lambda}} \frac{\lambda}{t_{\mathrm{P}}} e^{-\frac{\lambda}{t_{\mathrm{P}}} t} \tag{12}$$

with λ a constant. We first focus on getting an upper bound for t_{P} in the exponential. The upper bound for all the parent masses is given by the squared mass of the root vertex, t_{root} (top of the tree).

Next, we look for the biggest lower bound on the squared mass t in the exponential of Equation 12. We consider a fully unbalanced tree as the topology with the smallest per level values of t and we bound each level in a bottom up approach (singletons are at the bottom). For each internal node, t has to be greater than t_{cut} (t_{cut} is a threshold below which binary splittings stop in Ginkgo). Thus, for each element, we find the smallest parent invariant squared mass t_{Pi} that is above the threshold t_{cut}, else set this value to t_{cut} and save it to a list named t_{min}, that we sort. Maximizing the number of nodes per level (having a full binary tree) minimizes the values of t. We start at the first level, taking the first $N//2$ entries of t_{min} and adding them to our candidate minimum list, t_{bound} (N is the number of elements to cluster). For the second level, we take t_p^0 as the minimum value in t_{min} that is greater than t_{cut} and bound t by $[\tilde{t}(i\%2) + t_p^0(i//2)](j//2)$ with $i = 3$ and $j = N\%2 + N//2$, where $\tilde{t}$ is the minimum invariant squared mass among all the leaves. The reason is that $(j//2)$ bounds the total possible number of nodes at current level and i the minimum number of elements of the tree branch below a given vertex at current level. We calculate this bound level by level, maximizing the possible number of nodes at each level. This is described in Algorithm 2.

Finally, to get a bound for t_{P} (denoted as $\tilde{t}_P$) in the denominator of Equation 12 we want the minimum possible values. Here we consider two options for a heuristic:

- Admissible heuristic: Thus we take $\tilde{t}_P = t_{bound} + \tilde{t}$ (given that the parent of any internal node contains at least one more element) except when $t_{bound} < \max\{t_i\}_{i=0}^N$ where we just keep $t_{\mathrm{P}} = t_{bound}$ (see [29] for more details).

- h1: $\tilde{t}_P = t_{bound} + 2\, t_p^0$. Even though we do not have a proof for h1 to be admissible, we gained confidence about it from checking it on a dataset of 5000 Ginkgo jets with up to 9 elements (compared to the exact trellis solution for the MAP tree), as well as the fact that exact A* with h1 agrees with the exact trellis within 6 significant figures (see Figure 5).

Algorithm 2 Lower Bound on t in Equation 12

1: **function** LOWERBOUND(t_i, t_{min}, N)
2:   **Input:** Elements invariant mass squared: t_i, minimum value in t_{min} that is greater than t_{cut}: t_p^0, list with minimum parent mass squared above t_{cut} for each element: t_{min}, and number of elements: N.
3:   **Output:** List that bounds t: t_{bound}.
4:   ▷ Set initial variables
5:   $m^2 = sort([t_i \text{ for i in N}])$
6:   $t_{bound} = sort(t_{min})[0 : N//2]$
7:   $i = 3$
8:   $j = N\%2 + N//2$
9:   **do**
10:     $t_{bound}+ = [m^2[0](i\%2) + t_p^0(i//2)](j//2))$
11:     $j = j\%2 + j//2$
12:     $i+ = 1$
13:   **while** $len(t_{bound}) < N - 2$

Putting it all together, the upper bound for Equation 12 is

$$\sum \log f^{bound} = [-\log(1 - e^{-\lambda}) + \log(\lambda)] - \sum_{i=0}^{N-2} \left(\log(\tilde{t}_P) + \lambda * \frac{t_{bound}^i}{t_{root}} \right)$$

A.7.2 Leaves

We want to find the smallest possible upper bound to

$$f(t|\lambda, t_{\mathrm{P}}) = \frac{1}{1 - e^{-\lambda}} \left(1 - e^{-\frac{\lambda}{t_{\mathrm{P}}} t_{\mathrm{cut}}} \right) \tag{13}$$

with λ and t_{cut} constants.

We want the maximum possible value of t_{P} while still getting an upper bound in Equation 13. A parent t_{P} consists of a pairing of two elements (leaves). We find and sort the same list t_{min} as we did for the internal nodes. We do not know the order in which the two children are sampled, and the minimum value happens when each child is sampled last, $t_{\mathrm{Pi}} \rightarrow (\sqrt{t_{\mathrm{Pi}}} - \sqrt{t_i})^2$ (see [29] for more details). We take the minimum value greater than t_{cut} in t_{min} and refer to it as t_{Pi}^0. Thus we get $t_{\mathrm{Pi}} = (\sqrt{t_{\mathrm{Pi}}^0} - \sqrt{t_i})^2$. For the element with greatest invariant squared mass t_i we just keep $t_{\mathrm{Pi}} = t_{\mathrm{Pi}}^0$. Finally, sum the likelihood over each element i.

A.8 Dasgupta's Cost

Objective 2. (Dasgupta's Cost)[28] *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:*

$$\psi(X_i, X_j) = (|X_i| + |X_j|) \sum_{x_i, x_j \in X_i \times X_j} w_{ij} \tag{14}$$

This is equivalent to the cut-cost definition of Dasgupta's cost with the restriction to binary trees [28].

We are given the similarity graph, $\mathcal{W}(X)$, where the nodes of the graph refer to the dataset X and edges (denoted $E_{\mathcal{W}(X)}$), $w_{i,j}$ give similarity between points $x_i, x_j \in X$. Given a subset $X' \subseteq X$, we hope to design a heuristic $h(\cdot)$ such that $h(X') \leq \min_{H \in \mathbb{H}(X')} J(H)$.

We can think about Dasgupta's cost as a sum over weighted edges in the graph $E_{\mathcal{W}(X)}$ with each edge's similarity multiplied by the number of descendant nodes. This leads to a heuristic in which the number of descendants is lower-bounded by having a single descendant:

$$h_{\mathsf{d1}}(X) = \sum_{(i,j) \in E_{\mathcal{W}(X)}} w_{i,j} \tag{15}$$

Proposition 2. (Admissible Heuristic for Dasgupta) *Equation 15 is an admissible heuristic for Dasgupta's cost, that is $h_{\mathsf{d1}}(X) \leq J_{\mathsf{Dasgupta}}(X)$.*

Proof. We wish to prove that $\mathbb{H}_{dasgupta}(X) <= \operatorname{argmin}_{H \in \mathbb{H}(X)} \phi_{dasgupta}(H)$. $\square$

Note that in every hierarchical clustering, every element is eventually separated (at the leaves), thus every edge eventually crosses a cut in every tree, thus $\forall H \in \mathbb{H}(X)$,

$$\sum_{X_L, X_R \in \mathsf{sibs}(H)} \sum_{x_i, x_j \in X_L \times X_R} w_{ij} = \sum_{x_i, x_j \in X} w_{ij} \tag{16}$$

Since

$$\forall X, |X| \in \mathbb{Z}^+$$

We have

$$\phi(H) = \sum_{X_L, X_R \in \mathsf{sibs}(H)} \psi(X_L, X_R)$$

$$= \sum_{X_L, X_R \in \mathsf{sibs}(H)} ((|X_L| + |X_R|) \sum_{x_i, x_j \in X_L \times X_R} w_{ij})$$

$$>= \sum_{X_L, X_R \in \mathsf{sibs}(H)} \sum_{x_i, x_j \in X_L \times X_R} w_{ij}$$

$$= \sum_{x_i, x_j \in X} w_{ij} = \mathbb{H}_{Dasgupta}(X)$$