Exact and Approximate Hierarchical Clustering Using A*

Craig S. Greenberg[*,1] **Sebastian Macaluso**[*,2] **Nicholas Monath**[*,3]
Avinava Dubey[4] **Patrick Flaherty**[5] **Manzil Zaheer**[4] **Amr Ahmed**[4] **Kyle Cranmer**[2] **Andrew McCallum**[3]

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

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 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). Empirically we 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 [43], personalization [48], entity resolution [30, 37, 45], and jet physics [9, 11, 22, 25]. While much work has focused on approximation methods for relatively large datasets [3, 23, 24, 26, 34, 40, 47], there are also important use cases of hierarchical clustering that demand exact or high-quality approximations [32]. This paper focuses on one such application in jet physics: inferring *jet structures*.[1]

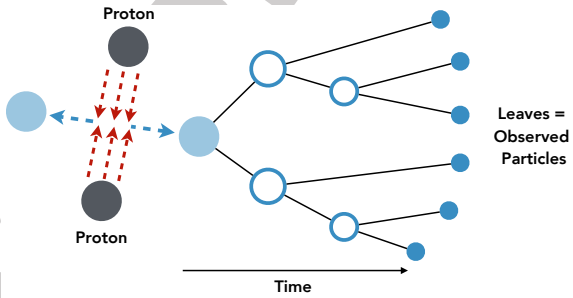

Figure 1: **Jets as hierarchical clustering**. Schematic representation of a process producing a jet at the Large Hadron Collider at CERN. Constituents of the incoming protons interact and produce two new particles (solid light blue). Each of them goes through successive binary splittings until giving 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.

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 shower-

[*]The first three authors contributed equally to this work.

[1]Note that we use jet physics as an example where *exact* clustering methods are needed and data sets are small enough for exact methods to be applied. Other work has shown the importance of exact methods for small data sets, including in clinical medicine/genomics[31, 32]. We emphasize that our approach applies to a variety of cost functions (Defn. 2), such as the hierarchical correlation clustering cost used for cancer genomics[31].

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

ing 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, typically with 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., [32], have limited scalability. It is standard to use agglomerative clustering techniques, which are greedy [10]. 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 state-of-the art results using deep learning methods that do not rely on hierarchical clusterings (they only take the leaves as input) [8] are 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 (and therefore would close the gap between traditional and deep learning classifiers in this context). Thus, exact and high-quality approximations in this context would be highly beneficial for data analyses in experimental particle physics.

Traditional algorithms, such as greedy approximations can lead 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, Daume III [21], uses A* to find MAP solutions to Dirichlet Process Mixture Models without bounds on the time and space complexity of the algorithm.

In this paper, we theoretically and empircally analyze our the following contributions:

Theoretical Contributions:

- **A* Datastructure**. Inspired by [31, 32], 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 (§3.2).
- **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).

Empirical Contributions:

- **Jet Physics**. We also demonstrate empirically that A* can find exact solutions to larger jet physics datasets than previous work [32], 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 [36] (§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. $X_j \bigcup X_k = X_i$.*

Consistent with our application in jet physics and previous work [32], we limit our consideration to hierarchical clusterings with binary branching factor.[2]

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).

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:*

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

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

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*

[2] Binary trees encode at least as many *tree consistent partitions* as multifurcating / n-ary trees [4, 32] and for well known cost functions, such as Dasgupta's cost, provide trees of lower cost than multifurcating / n-ary trees [20].

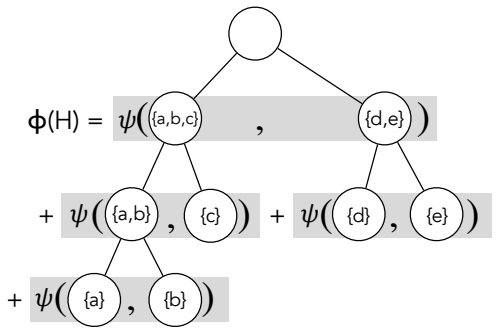

Figure 2: **Family of Hierarchical Clustering Cost Functions**. We considers costs that decompose as the sum of a cost term computed for each pair of sibling nodes.

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 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 represents the A* state space and frontier, and then show how to perform A* search using this data structure to find the lowest cost clustering.

3.1 HIERARCHICAL CLUSTERING AS SEARCH

Many approaches to hierarchical clustering 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, and, in those cases, hierarchical clustering can be seen as a combinatorial optimization problem. This view enables the use of search methods for hierarchical clustering, as search methods can often be applied to combinatorial optimization problems [1].

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. $\breve{H} \in \mathbb{H}(X)$.*

Each state is a hierarchical clustering over the set of points, X, built in a 'top-down' way. It includes a root node (all of X) and some internal nodes. Importantly, the state might not be a complete hierarchical (i.e., a goal state)– the leaves of the state might not be the data points of X (a requirement of a complete hierarchical clustering).

A* Search for Hierarchical Clustering 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.

A heuristic, $h(n)$, for a function, $f(n)$, is said to be *admissible* when the heuristic underestimates the function, i.e., $\forall n, h(n) \leq f(n)$.[3] When the heuristic $h(\cdot)$ is admissible, 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*[4].

Consider the following naïve A* approach for hierarchical clustering: (0) add the root node of a tree structure to the search frontier (i.e., the set of states A* is currently open to exploring), (1) visit the state on the frontier with the lowest cost according to the heuristic, and (2) add to the frontier (or update) the states neighboring the visited state (using the heuristic). Such a naïve search could require the frontier to include all possible (sub)-tree structures, which would require super exponential space. Such a space requirement renders the naïve approach intractable. In the following sections, we describe how the trellis can be used to implement a more efficient approach.

3.2 A* ON THE CLUSTER TRELLIS

We propose an alternative representation of the search space and frontier that utilizes a trellis data structure to provide a compact encoding of partial tree structures. 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, [32], 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 clusters, and edges in the trellis

[3] This is the case when attempting to minimize $f(n)$; when maximizing $f(n)$, heuristic $h(n)$ is admissible when it *overestimates* $f(n)$, i.e., $\forall n, h(n) \geq f(n)$.

[4] 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.

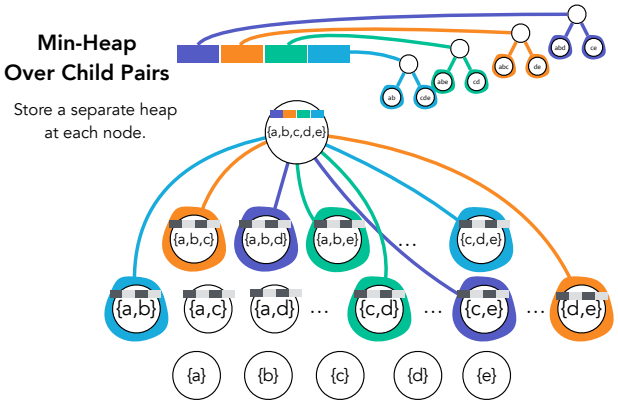

Figure 3: **Nested Min Heaps** The A* search space is compactly encoded in the trellis. Each node stores a min-heap ordering pairs of its children. Each pair encodes a two partition (e.g. split) of its parents points.

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) [32] *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).*

We note the distinction between two types of trellis structures: a"full" trellis, one that contains all possible subsets of the dataset (i.e., $\mathbb{T} = \mathbb{P}(X)$) and a"sparse", i.e., contains only a subset of all possible subsets ($\mathbb{T} \subset \mathbb{P}(X)$).

We will now describe how we modify the trellis structure to run A* in less than super-exponential time and space. We detail how the full trellis provides exact solutions and sparse trellis structures provide the best approximate solution in a restricted search space. To run A* on trellis $\mathbb{T}$, we modify the data structure to 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 $X_i[\Pi]$. Additionally, we modify each node to store the value of the lowest cost split at each node, $X[\Phi]$, as well as a pointer to children associated with the best split, $X[\Xi] = \{X_\ell, X_r\}$.

At a high level, A* using the trellis is executed by (1) finding the state in the search frontier that minimizes $f(\cdot)$ by traversing the min-heaps stored trellis in a top-down manner (2) exploring the neighbors of the state by initializing the min-heaps at the leaves of the current state's tree, and then updating the $f(\cdot)$ values on the min-heaps on the leaf-to-root paths of the current state's tree.

First, let's consider how to find the current best state on the search frontier (i.e., the state minimizing $f(\cdot)$). The starting state of the search is the root node (of both the output tree structure and the trellis), which corresponds to the full dataset, X. We compactly encode a partial hierarchical clustering as a set of paths through the trellis structure starting with the root. 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 $\mathsf{lvs}(H) = \{X \mid X \in H, \nexists X' \in H, X' \subset X\}$. In words, we start at the root node, add 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, select the pair of children (grandchildren of the root) from their min-heaps to add to the tree structure. We proceed in this way, adding descendants until it reaches nodes that do not yet have min-heaps initialized.

Now, let's consider how the neighbors of the current best state are explored and the search frontier is updated. Each node's min heap, $X_i[\Pi]$, stores five-element tuples:

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

Figure 3 illustrates a trellis compactly encoding the search space using min heaps, where four of the fifteen child pairs on the root node's heap are depicted in different colors.

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:

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

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.

With these definitions, let's consider the execution of A* using the trellis in more detail. Recall that the trellis compactly encodes the frontier by storing a min-heap at each explored node. Each heap contains edges to all the children (2-splittings of that node), and the frontier is the set

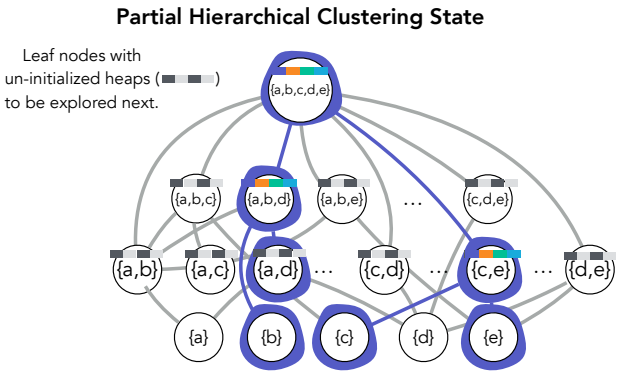

Figure 4: **State Space** A state (i.e., a partial hierarchical clustering) is compactly encoded as paths in the trellis structured, discovered by following the root-to-leaf pointers.

of partial hierarchical clusterings with explored nodes/min-heap edges as internal nodes. (The min-heaps and the nested structure are described in Section 3.2.) Search works by: (0) the root node is added frontier; (1) Eq. 2 is used to find the current best state, a partial hierarchical clustering, on the search frontier (Alg. 1:5-7); (2) If this state is a goal state and the h_{LR} value at the top of the min heap is 0, then return this structure (Alg. 1:8-10); (3) 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) (Alg. 1:11-17); (4) 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 (Alg. 1:18-31). Crucially, Step 1 is implemented using the trellis following the pointers stored in each node's min-heap. Step 4 is the book-keeping required to use the trellis to achieve the reduced running time.

This process is illustrated in Figure 4, where the set of purple nodes represent a state, and the grey nodes/edges correspond to the remainder of the trellis. Note that this is a state in the A* search trajectory, while the nodes/edges correspond to hierarchical clustering nodes/edges, not states / transitions between states. The hierarchical clustering in Figure 4 is partial, not complete, because the leaves of the hierarchical clustering include a node with two elements, a,d. One state H is reachable from another H' if H can be obtained by splitting the leaves of H'. See Algorithm 1 for pseudocode.

3.3 THEORETICAL ANALYSIS

First, we show that A* using the trellis (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 $H^\star = \text{argmin}_{H \in \mathbb{H}(\mathbb{T})} \phi(H)$ where $\mathbb{H}(\mathbb{T})$ is the set of all trees represented by the trellis.*

See Appendix §A.1 for proof.

Corollary 1. (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.

Proposition 1. (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.

Theorem 2. (Time Complexity) *For trellis, $\mathbb{T}$ and dataset, X, Algorithm 1 finds the lowest cost hierarchical clustering of X in time $O(|\{ \text{CH}(X, \mathbb{T}) \mid X \in \mathbb{T}\}|)$. 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.

Proposition 2. (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

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 proposes 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 either in a top-down or bottom-up manner. However, this structure can be updated during run-time using

Algorithm 1 A* Hierarchical Clustering Search

function SEARCH($\mathbb{T}$, X, $\mathbb{H}$)
  Input: A trellis structure $\mathbb{T}$ and dataset X, a heuristic function $\mathbb{H}$
  Output: Lowest cost tree represented in $\mathbb{T}$
  do
    ▷ Get state from trellis (Eq. 2)
    $H_\Xi(X) = \{X\} \bigcup_{X_c \in X[\Xi]} H_\Xi(X_c)$
    $f_{\text{at root}}, \_, h_{\text{at root}}, \_, \_ \leftarrow X[\Pi]$.peek()
    ▷ At goal state?
    if $h_{\text{at root}} = 0$ **and** $|\text{lvs}(H_\Xi(X))| = |X|$ **then**
      return $f_{\text{at root}},\ H_\Xi(X)$
    ▷ Explore new leaves
    for X_i **in** lvs$(H_\Xi(X))$ **do**
      for $X_L,\ X_R$ **in** CHILDREN(X_i) **do**
        Define g_{LR} according to Eq. 4
        Define h_{LR} according to Eq. 5
        $f_{LR} \leftarrow g_{LR} + h_{LR}$
        $X_i[\Pi]$.enqueue$((f_{LR}, g_{LR}, h_{LR}, X_L,\ X_R))$
    ▷ Update each node in the tree's min heap
    for X_i **in** $H_\Xi(X)$ **from leaves to root do**
      $\_, \_, \_, X_L,\ X_R \leftarrow X_i[\Pi]$.pop()
      Define $g \leftarrow g_{LR}$ according to Eq. 4
      $h \leftarrow 0$
      for X_c **in** $[X_L, X_R]$, **do**
        if $X_c[\Pi]$ **is defined then**
          $\_, g_c, h_c, \_,\ \_ \leftarrow X_c[\Pi]$.peek()
          $g \leftarrow g + g_c$
          $h \leftarrow h + h_c$
        else
          $h \leftarrow h + \mathbb{H}(X_c)$
      ▷ Update the f value of split X_L, X_R
      $X_i[\Pi]$.enqueue$((g + h, g, h, X_L, X_R))$
  while True

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. We can 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 the trellis is full.

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., [9, 11, 22, 25]), 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 [19]. Each cluster 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 [19] 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,

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

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

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 [19] for more details),

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

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 [19] 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 $\mathsf{h1}(\cdot)$.

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 [6] for more details), which significantly improved the performance of beam search. We also compare to the Monte-Carlo Tree Search (MCTS) algorithm for hierarchical clusterings introduced in [7]. 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 algorithm introduced in [32].

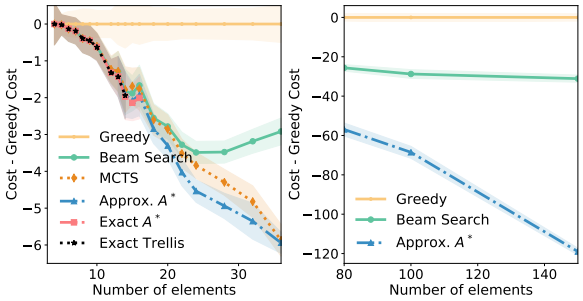

Figure 5: **Jet Physics Results.** Cost (Neg. log. likelihood) for the MAP hierarchy of each algorithm on Ginkgo datasets minus the cost for greedy (lower values are better solutions). We see that exact and approx. A* greatly improve over the benchmarks. Though MCTS provides a strong baseline for small datasets where it is feasible to implement it (left), A* still shows an improvement.

Approximate A*. We design the approximate A*-based method following the details described in Section 4. We first initialize the sparse trellis with the full set of beam search hierarchies. Next, we use a top K-based sampling and the iterative construction procedure to extend the search space.

Results. 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 the 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 as a power law of the number of elements. 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

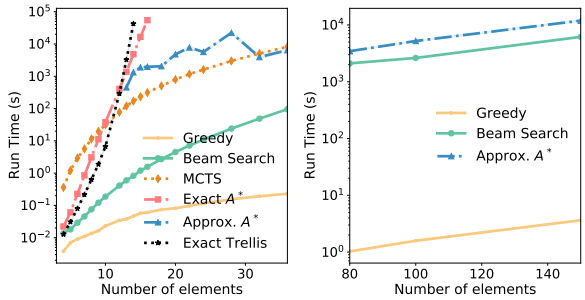

Figure 6: **Running Times.** Empirical running time on an Intel Xeon Platinum 8268 24C 2.9GHz Processor. The A*-based method becomes faster than the exact trellis one for data with 12 or more elements. Approx. A* is much faster than exact algorithms while still significantly improving over the baselines. 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 runtime.

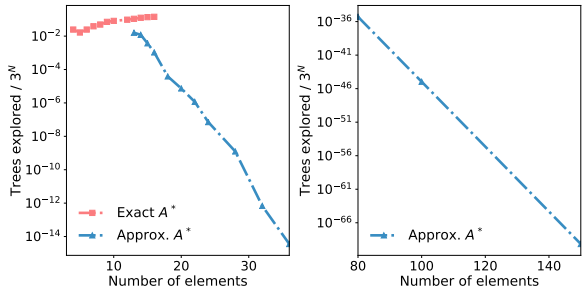

Figure 7: **Search Space Exploration.** Number of hierarchies explored divided by the exact trellis time complexity, i.e 3^N. We see that the A*-based method is considerably more efficient than the trellis, which, in turn, is super-exponentially more efficient than an explicit enumeration of clusterings.

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 $\mathcal{O}(10^3)$ out of the $\mathcal{O}(10^{185})$ possible hierarchies.

6 EXPERIMENTS ON CLUSTERING BENCHMARKS

In this section, we analyze the performance of A* on several commonly used hierarchical clustering benchmark datasets [36, 38]: **Speaker**, i-vector speaker recordings, ground truth clusters refer each unique speaker [33]; **CovType**, a dataset of forest cover types; **ALOI**, color histogram features of 3D objects, ground truth clusters refer to each object type [27]; **ILSVRC**, image features from InceptionV3 [44] of the ImageNet ILSVRC 2012 dataset [42]. We use cosine similarity as the pairwise similarity between points, as it is known to be meaningful for each of the datasets [38], and hierarchical correlation clustering as the cost.

Hierarchical Correlation Clustering. Following [32], we consider a hierarchical version of well-known flat clustering objective, correlation clustering [2]. 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:

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

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 heuristic for this objective by using the sum of the of the positive edges contained in the dataset:

$$h_{\mathsf{cc}}(X) = \sum_{\substack{x_i, x_j \in X \times X, \\ i<j}} w_{ij} \mathbb{I}[w_{ij} > 0] \tag{9}$$

Proposition 3. (Admissible Heuristic for Hierarchical Correlation Clustering) *Equation 9 is an admisslbe heuristic for Hierarchical Correlation Clustering cost, that is* $h_{cc}(X) \leq \phi_{HCC}(X)$.

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 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

Methods: Most similar to our approach, Daume III [21] uses an A*-based approach to find MAP solutions to Dirich-

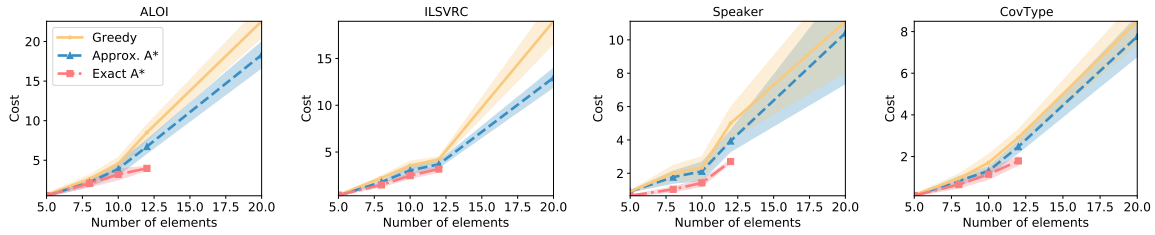

Figure 8: **Hierarchical Correlation Clustering on Benchmark Data**. On randomly sampled subsets small enough for us to run the exact A* approach, we report the hierarchical correlation clustering cost (lower is better) for clustering benchmarks. We observe that the approximate A* method is able to achieve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

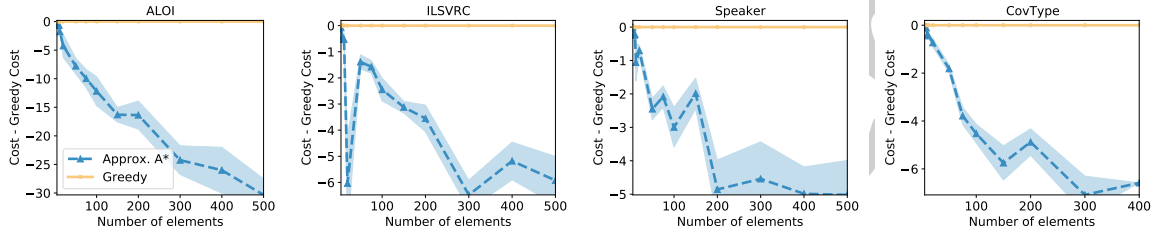

Figure 9: **Improvement over Greedy on Larger Benchmark Datasets** . We report the improvement (cost, lower is better) of the approximate A* method over the greedy baseline.

let 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 [5, 35, 28, 6], 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 [36, 38, 46], 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 [29, 41], as have, branch-and-bound based methods [18]. 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 structures to support gradient-based optimization [39, 12]. Finally, we note that much recent work, [13, 14, 15, 16, 17, 41], has been devoted to Dasgupta's cost for hierarchical clustering [20]. 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.

Data Structure: The general idea of a cluster trellis data structure has been used in two previous works: [31] for flat and [32] for hierarchical clustering. In each, nodes correspond to the powerset of the set of points. These works [31, 32] use the trellis to design dynamic programming algorithms to find the exact lowest cost clustering. In contrast, the trellis in this paper, performs exact search with A*, leveraging the critical innovation of nesting min-heaps in the trellis to compactly encode the A* frontier and enabling A* search to run with improved time/space complexity.

8 CONCLUSION

In this paper, 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 apply A* to flat clustering costs and probabilistic models in Bayesian non-parametrics.

ACKNOWLEDGEMENTS

We thank Johann Brehmer for sharing his code and helping us run the MCTS algorithm for hierarchical clustering of Ginkgo jets on our benchmark datasets. Kyle Cranmer and Sebastian Macaluso are supported by the National Science Foundation under the awards ACI-1450310 and OAC-1836650 and by the Moore-Sloan data science environment at NYU. Patrick Flaherty is supported in part by NSF HDR TRIPODS award 1934846. Andrew McCallum and Nicholas Monath are supported in part by the Center for Data Science and the Center for Intelligent Information Retrieval, and in part by the National Science Foundation under Grant No. NSF-1763618. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor.

References

[1] Emile Aarts, Emile HL Aarts, and Jan Karel Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 2004.

[3] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. Affinity clustering: Hierarchical clustering at scale. In *NeurIPS*, 2017.

[4] C Blundell, YW Teh, and KA Heller. Bayesian rose trees. In *UAI*, 2010.

[5] Alexandre Bouchard-Côté, Sriram Sankararaman, and Michael I Jordan. Phylogenetic inference via sequential monte carlo. *Systematic biology*, 2012.

[6] Levi Boyles and Max Welling. The time-marginalized coalescent prior for hierarchical clustering. In *NeurIPS*, 2012.

[7] Johann Brehmer, Sebastian Macaluso, Duccio Pappadopulo, and Kyle Cranmer. Hierarchical clustering in particle physics through reinforcement learning. In *Machine Learning and the Physical Sciences workshop at NeurIPS*, 2020.

[8] Anja Butter et al. The Machine Learning Landscape of Top Taggers. 2019.

[9] Matteo Cacciari, Gavin P Salam, and Gregory Soyez. The anti-k_t jet clustering algorithm. *JHEP*, 2008.

[10] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet User Manual. *Eur. Phys. J. C*, 2012.

[11] S Catani, Yuri L Dokshitzer, M H Seymour, and B R Webber. Longitudinally invariant k_t clustering algorithms for hadron hadron collisions. *Nucl. Phys. B*, 1993.

[12] Ines Chami, Albert Gu, Vaggos Chatziafratis, and Christopher Ré. From trees to continuous embeddings and back: Hyperbolic hierarchical clustering. *NeurIPS*, 2020.

[13] Moses Charikar and Vaggos Chatziafratis. Approximate hierarchical clustering via sparsest cut and spreading metrics. In *SODA*, 2017.

[14] Moses Charikar, Vaggos Chatziafratis, and Rad Niazadeh. Hierarchical clustering better than average-linkage. In *SODA*, 2019.

[15] Vaggos Chatziafratis, Rad Niazadeh, and Moses Charikar. Hierarchical clustering with structural constraints. *ICML*, 2018.

[16] Vincent Cohen-Addad, Varun Kanade, and Frederik Mallmann-Trenn. Hierarchical clustering beyond the worst-case. In *NeurIPS*, 2017.

[17] Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. Hierarchical clustering: Objective functions and algorithms. *JACM*, 2019.

[18] Carlos Cotta and Pablo Moscato. A memetic-aided approach to hierarchical clustering from distance matrices: application to gene expression clustering and phylogeny. *Biosystems*, 2003.

[19] Kyle Cranmer, Sebastian Macaluso, and Duccio Pappadopulo. Toy Generative Model for Jets, 2019. URL `https://github.com/SebastianMacaluso/ToyJetsShower/blob/master/notes/toyshower_v4.pdf`.

[20] Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *STOC*, 2016.

[21] Hal Daume III. Fast search for dirichlet process mixture models. In *AISTATS*, 2007.

[22] Yuri L Dokshitzer, G D Leder, S Moretti, and B R Webber. Better jet clustering algorithms. *JHEP*, 1997.

[23] Avinava Dubey, Qirong Ho, Sinead Williamson, and Eric P Xing. Dependent nonparametric trees for dynamic hierarchical clustering. *NeurIPS*, 2014.

[24] Kumar Avinava Dubey, Michael Zhang, Eric Xing, and Sinead Williamson. Distributed, partially collapsed mcmc for bayesian nonparametrics. In *AISTATS*, 2020.

[25] Stephen D Ellis and Davison E Soper. Successive combination jet algorithm for hadron collisions. *Phys. Rev. D*, 1993.

[26] Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Bico: Birch meets coresets for k-means clustering. In *European Symposium on Algorithms*. Springer, 2013.

[27] Jan-Mark Geusebroek, Gertjan J. Burghouts, and Arnold W. M. Smeulders. The amsterdam library of object images. In *IJCV*, 2005.

[28] Zoubin Ghahramani, Michael I Jordan, and Ryan P Adams. Tree-structured stick breaking for hierarchical data. In *NeurIPS*, 2010.

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

[30] Spence Green, Nicholas Andrews, Matthew R. Gormley, Mark Dredze, and Christopher D. Manning. Entity clustering across languages. In *NAACL-HLT*, 2012.

[31] Craig Greenberg, Nicholas Monath, Ari Kobren, Patrick Flaherty, Andrew McGregor, and Andrew McCallum. Compact representation of uncertainty in clustering. In *NeurIPS*. 2018.

[32] Craig Greenberg, Sebastian Macaluso, Nicholas Monath, Ji Ah Lee, Patrick Flaherty, Kyle Cranmer, Andrew McGregor, and Andrew McCallum. Cluster trellis: Data structures & algorithms for exact inference in hierarchical clustering. In *AISTATS*, 2021.

[33] Craig S Greenberg, Désiré Bansé, George R Doddington, Daniel Garcia-Romero, John J Godfrey, Tomi Kinnunen, Alvin F Martin, Alan McCree, Mark Przybocki, and Douglas A Reynolds. The nist 2014 speaker recognition i-vector machine learning challenge. In *Odyssey: The Speaker and Language Recognition Workshop*.

[34] Zhiting Hu, Ho Qirong, Avinava Dubey, and Eric Xing. Large-scale distributed dependent nonparametric trees. *ICML*, 2015.

[35] David A Knowles and Zoubin Ghahramani. Pitman-yor diffusion trees. In *UAI*, 2011.

[36] Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A hierarchical algorithm for extreme clustering. In *KDD*, 2017.

[37] Michael Levin, Stefan Krawczyk, Steven Bethard, and Dan Jurafsky. Citation-based bootstrapping for large-scale author disambiguation. *JASIST*, 2012.

[38] Nicholas Monath, Ari Kobren, Akshay Krishnamurthy, Michael R Glass, and Andrew McCallum. Scalable hierarchical clustering with tree grafting. In *KDD*, 2019.

[39] Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. *KDD*, 2019.

[40] Stanislav Naumov, Grigory Yaroslavtsev, and Dmitrii Avdiukhin. Objective-based hierarchical clustering of deep embedding vectors. *AAAI*, 2021.

[41] Aurko Roy and Sebastian Pokutta. Hierarchical clustering via spreading metrics. *NeurIPS*, 2016.

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. In *IJCV*, 2015.

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

[44] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[45] Shikhar Vashishth, Prince Jain, and Partha Talukdar. Cesi: Canonicalizing open knowledge bases using embeddings and side information. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1317–1327. International World Wide Web Conferences Steering Committee, 2018.

[46] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 1996.

[47] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1997.

[48] Yuchen Zhang, Amr Ahmed, Vanja Josifovski, and Alexander Smola. Taxonomy discovery for personalized recommendation. In *WSDM*, 2014.