EPJ Web of Conferences **251**, 03059 (2021) https://doi.org/10.1051/epjconf/202125103059
CHEP 2021

Reframing Jet Physics with New Computational Methods

Kyle Cranmer[1], Matthew Drnevich[1], Sebastian Macaluso[1,*], and Duccio Pappadopulo

[1]New York University

Abstract. We reframe common tasks in jet physics in probabilistic terms, including jet reconstruction, Monte Carlo tuning, matrix element – parton shower matching for large jet multiplicity, and efficient event generation of jets in complex, signal-like regions of phase space. We also introduce `Ginkgo`, a simplified, generative model for jets, that facilitates research into these tasks with techniques from statistics, machine learning, and combinatorial optimization. We also review some of the recent research in this direction that has been enabled with `Ginkgo`. We show how probabilistic programming can be used to efficiently sample the showering process, how a novel trellis algorithm can be used to efficiently marginalize over the enormous number of clustering histories for the same observed particles, and how the dynamic programming and reinforcement learning can be used to find the maximum likelihood clusterinng in this enormous search space. This work builds bridges with work in hierarchical clustering, statistics, combinatorial optmization, and reinforcement learning.

1 Introduction

Jets are the most copiously produced objects at physics colliders, such as the Large Hadron Collider (LHC) or the Relativistic Heavy Ion Collider (RHIC), and the subject of intense experimental and theoretical study. Improvements to our understanding and treatment of jets can have significant impact on the physics program of the LHC and RHIC, as well as future colliders; however, various computational bottlenecks appear in this quest. Below we will discuss a few areas where such computational bottlenecks appear and identify emerging computational techniques that may be able to address them. We hope that this may challenge some assumptions about the computational demands of simulation, reconstruction, and analysis of collider physics data when jets are involved.

1.1 Reframing jet physics in probabilistic terms

Monte Carlo event generators (e.g. simulators like PYTHIA [1], Herwig [2], and Sherpa [3]) encode a physics model for the fragmentation and hadronization of quarks and gluons produced at colliders. In statistical and machine learning language, they are generative models for jets. Following the notation of Ref. [4, 5], we denote the parameters of the (Monte Carlo) simulation θ, the observable output of the simulator x, and latent variables (aka Monte Carlo truth record or showering history) z. The simulators typically evolve the latent state sequentially as a Markov process and model the physics of each splitting, clustering, etc. In the original parton showers based on successive $1 \to 2$ splittings, the joint likelihood for the parton shower can be expressed as:

$$p(x, z|\theta) = p(x|z_{\text{leaves}}) \prod_{s \in \text{splittings}} p(z_{s,L}, z_{s,R}|z_{s,P}) \,, \tag{1}$$

*e-mail: seb.macaluso@nyu.edu

where $z_{s,P}$, $z_{s,L}$, and $z_{s,R}$, and are respectively the data needed to encode the state of the parent and left and right children for the s^{th} splitting and z_{leaves} are the terminal leaves of the showering process. The hadronization and detector simulation fit in this framing as well, but we do not discuss it explicitly in this work.

We find it elucidating to reframe the following concepts in jet physics in probabilistic terms:

- Joint likelihood for latent shower and observed constituents $p(x, z|\theta)$
- Marginal likelihood for observed constituents $p(x|\theta) = \int dz\ p(x, z|\theta)$
- Maximum likelihood showering history $\hat{z} = \text{argmax}_z\ p(x|z, \theta)$
- Maximum likelihood parameters for the model $\hat{\theta} = \text{argmax}_\theta\ p(x|\theta) = \text{argmax}_\theta \int dz\ p(x, z|\theta)$
- Posterior distribution on showering histories $p(z|x, \theta)$

A few challenges present themselves in this framing of jet physics.

First of all, the joint likelihood $p(x, z|\theta)$ and the likelihood of individual splittings $p(z_{s,L}, z_{s,R}|z_{s,P})$ is not exposed in a way that is convient to access. The joint likelihood corresponds to what is coded in PYTHIA [1], Herwig [2], and Sherpa [3], but often in terms of accept-reject sampling and procedural code that does not explicitly expose the probabilities themselves. This motivates `Ginkgo`, which provides convenient access to these quantities in a simplified parton shower.

Secondly, the joint likelihood $p(x, z|\theta)$ is not immediately of interest to experimentalists since the (latent) showering history z is not observed. Quantities such as the marginal likleihood $p(x|\theta)$ and the maximum likelihood parameter $\hat{\theta}$ involve integration (sums) over all possible showering histories. The number of possible showering histories grows factorially with the number of jet constituents. This super-exponential growth in the number of showering histories is at the heart of many computational bottlenecks in jet physics, making the marginalization and maximization over the latent space z of showering histories typically intractable.

Next we will review some common tasks in jet physics framed in these probabilistic terms. We will identify the computational challenges and the potential for emerging computational techniques to address them. In Section2 we will describe `Ginkgo`'s simplified probabilistic model for the parton shower. Finally, we will review some of the recent research into these problems enabled with `Ginkgo`.

1.1.1 Jet clustering

Jet reconstruction can be thought of as estimating the latent state (showering history) z from the observed particles x. Traditionally, given a set of final state particles, jets are reconstructed using one of the generalized k_t clustering algorithms [6–9]. These algorithms sequentially cluster jet constituents by merging the closest pair based on the distance measure d_{ij}^{α},

$$d_{ij}^{\alpha} = \min(p_{ti}^{2\alpha}, p_{tj}^{2\alpha})\frac{\Delta R_{ij}^2}{R^2} \tag{2}$$

where ΔR_{ij} is the angular distance for the pair $\langle i, j\rangle$, R is a fixed value for the jet radius and $\alpha = \{-1, 0, 1\}$ specifies the anti-k_t, Cambridge/Aachen (C/A) and k_t algorithms respectively. This traditional approach doesn't make explicit reference to the probability model for the jet, but intuitively the k_t and C/A algorithms cluster together two constituents that are likely to have emerged from the same parent. [1] We can formulate the intuition for the k_t and C/A algorithms as saying that the distance measure $d_{i,j}^{\alpha}$ is monotonically decreasing with the splitting likelihood $p(z_{s,L}, z_{s,R}|z_{s,P})$.

[1] In contrast, the anti-k_t algorithm focuses more on having desirable global properties for the jets than reconstructing a physically motivated showering history.

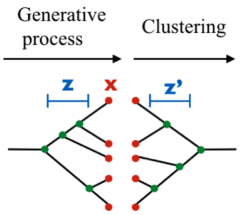

Figure 1. Schematic representation of the tree structure of a jet generated with Ginkgo and the resulting tree for some clustering algorithm. For a given algorithm, z labels the different variables that determine the latent structure of the tree. The tree leaves x are labeled in red and the inner nodes in green.

In the probabilistic language, the natural goal is to find the most likely clustering $\hat{z}$. In that light, the generalized-k_t algorithms are greedy algorithms for finding the maximum likelihood clustering. Greedy algorithms aren't guaranteed to find the maximum likelihood clustering because they don't consider the tree globally. More sophisticated algorithms like beam search, which are used widely in natural language processing, look more than one step ahead and consider multiple possible clusterings in memory as they proceed. They are guaranteed to recover jet clusterings that are at least as good as the greedy algorithm, and can be expected to improve upon them. But to do this, one needs a way to score the combination of multiple clusterings. It's not clear how one would combine the distance measure $d^{\alpha}_{i,j}$ for two splittings, but there is an natural rule for combining the splitting likelihood $p(z_{s,L}, z_{s,R}|z_{s,P})$ (i.e. their product). This is one of the advantages of the probabilistic formulation: it allows us to recast the objective of greedy clustering algorithms like k_t and extend them to more sophisticated algorithms.

Jet clustering can also be framed as a hierarchical clustering task. In that framing, the generalized-k_t algorithms are considered (bottom-up) hierarchical agglomerative clustering (HAC) algorithms.

Later we will review the research in jet clustering enabled by `Ginkgo` including a novel trellis data structure and dynamic programming model, a novel A^* search algorithm that makes use of the trellis data structure and an admissible heuristic, and a few reinforcement learning algorithms including Monte Carlo Tree Search and Behavioral Cloning.

1.1.2 Tuning the parameters of the shower model

Monte Carlo tuning can be thought of as estimating θ given a dataset of $\{x_i\}$. Ideally, if we wanted to fit (tune) the parameters θ of PYTHIA [1], Herwig [2], or Sherpa [3] (and we had infinite computing power), then we would compute the maximum likelihood $\hat{\theta}$ based on the high dimensional jet data x. Since we do not, we resort to tools like `Professor` [10], which compare projections of complicated events to individual variables (marginal distributions), which is blind to various forms of mismodelling in the high-dimensional structure of the jets. The marginalizaton over the latent space is implicit when forming the histograms of these individual differential distributions. The fact that tuning the generators is itself a bottleneck suppresses the motivation to add even more flexibility and parameters to the shower models, even if they might lead to more accurate description of the jets.

The first emerging technique in this direction is *likelihood-free inference* or *simulation-based inference* [4, 11, 12]. Recent progress in this direction includes likelihood-free inference methods [4, 5, 11, 12]. These methods approximate the intractable $p(x|\theta)$ using machine learning and bypass an explicit marginalization over the latent state z. The techniques can exploit the joint likelihood $p(x, z|\theta)$ if it is available. An implementation of these techniques for events simulated with Pythia was introduced in Refs. [13]. A closely related approach was outlined in Ref. [14]

An alternate approach to this problem would be to use probabilistic programming techniques to efficiently approximate the intractable integrals. The first prototypes of integrating probabilistic programming with the Monte Carlo generators (specifically Sherpa) was performed in Refs. [15, 16]

1.1.3 Event Generation for events with large jet multiplicity

The enormous number of possible showering histories is a bottleneck in the simulation of multijets events [17, 18] and shower deconstruction [19–21]. When implementing the CKKW-L matching algorithm [17, 18], parton final states need to be reweighted with the corresponding Sudakov form factors of each history, $p(x, z|\theta)$. The standard algorithm typically becomes infeasible for parton level configurations that exceed the complexity of $W/Z+$ 6 jet final state [22] due to the super-exponential growth in the number of clustering histories.

To ameliorate these bottlenecks we introduced in [23], a novel data structure and algorithms, called hierarchical cluster trellis, that can be used to efficiently represent the distribution over trees. The trellis can be used to compute the marginal likelihood $p(x|\theta)$ or the exact maximum likelihood showering history $\hat{z}$ in time and memory proportional to the significantly smaller powerset of the number of jet constituents, i.e. 2^N. In particular, we showed that the trellis allows us to perform these operations for larger values of N where the naive iteration over the $(2N-3)!!$ trees is impractical. Thus far the implementation is based on binary trees, $1 \to 2$ splittings; however, it is possible to extend the cluster trellis to consider $2 \to 3$ splittings required in the CKKW-L algorithm.

The trellis data structure also provides an efficient dynamic programming algorithm to find the maximum likelihood shower history $\hat{z}$, which provides a principled alternative to the generalized k_t algorithms, which are based on a greedy sequential clustering algorithm as described in Sec. 1.1.1.

1.1.4 Simulating jet backgrounds in signal-rich regions of phase space

Simulating sufficient numbers of multijet background events is a computational challenge due to the enormous rate of multijet events and their steeply falling spectra. The experimental collaborations have traditionally sliced the phase space into exclusive regions (eg. based on the p_T of the leading jet at parton level). This is an effective strategy for populating the tails of that distribution, but it is not effective for populating complicated phase space regions (eg. QCD events that satisfy the cut on a boosted top-tagger). Generating enough background Monte Carlo events in these signal-like regions of phase space is one of the computational challenges for the LHC and HL-LHC.

Denote passing an event selection cut with the indicator function $\mathbf{1}(x)$. In our probabilistic language, we are interested in efficiently sampling the showering histories $z \sim p(z|\mathbf{1}(x), \theta)$ so that we do not waste computing resources on the expensive detector simulation for events that won't satisfy the cuts. When the phase space regions are not aligned with parton level quantities, then we must perform importance sampling in the parton shower itself, and the ideal importance distribution would be the unfolded $p(z|\mathbf{1}(x), \theta)$, which is difficult to estimate when working with cuts based on complicated jet observables.

Recent developments in probabilistic programming systems offer a potential way to address these challenges. Probabilistic programming systems provide tools for inferring the latent state of a simulator based on some observations (e.g., $p(z|x, \theta)$), and they use the simulator directly during inference. As mentioned above, the `pyprob` probabilistic programming system was integrated with the Sherpa event generator via the `ppx` protocol [15, 16]. By instrumenting Sherpa with `ppx`, the `pyprob` system is able to bias the control flow of the event generator to perform advanced forms of importance sampling. In Refs. [15, 16] a large recurrent neural network learned an efficient importance sampling distribution $q(z|x)$; however, the target was τ lepton decay instead of jet physics. More recently, we have instrumented our `Ginkgo` generator with `pyprob`, which we will describe below.

2 Ginkgo: A simplified generative model for jets

At present, it is very hard to access the joint likelihood in state-of-the-art parton shower generators in full physics simulations, e.g. PYTHIA [1], Herwig [2], and Sherpa [3]. Also, typical implementations of parton showers involve sampling procedures that destroy the analytic control of the joint likelihood. Thus, to aid in machine learning research for jet physics, a python package for a toy generative model of a parton shower, called Ginkgo, was introduced in [24]. Ginkgo has a tractable joint likelihood, and is as simple and easy to describe as possible but at the same time captures essential ingredients of parton shower generators in full physics simulations. It also ensures permutation invariance and momentum conservation. Ginkgo was designed to enable implementations of *probabilistic programming, differentiable programming, dynamic programming* and *variational inference*. Within the analogy between jets and NLP, Ginkgo can be thought of as ground-truth parse trees with a known language model.

Ginkgo implements a recursive algorithm to generate a binary tree, where each node is represented by an energy-momentum vector and the leaves are the jet constituents. We want our model to represent the following features:

- Momentum conservation: the total momentum of the jet (root of the tree) is obtained from adding the momentum of all of its constituents.
- Running of the splitting scale: each splitting is characterized by a scale t that decreases when evolving down the tree from root to leaves (t is the invariant squared mass, $t = m^2$).

We also want our model to lead to a natural analogue of the generalized k_t clustering algorithms [6–9] for the generated jets. These algorithms are characterized by

- Permutation invariance: the jet momentum should be invariant with respect to the order in which we cluster its constituents.
- Distance measure: the angular separation between two jet constituents is typically used as a distance measure among them. In particular, traditional jet clustering algorithms are based on a measure given by $d_{ij} \propto \Delta R_{ij}^2$ where ΔR_{ij} is the angular separation between two particles.

We build our model as follows. During the generative process, starting from the root of the tree, each parent node is split, generating a left (L) and a right (R) child. At each splitting we sample squared invariant masses for the children, t_L, t_R from a decaying exponential. We require the constraint $\sqrt{t_L} + \sqrt{t_R} < \sqrt{t_P}$, where $\sqrt{t_P}$ is the parent mass. Then we implement a 2-body decay in the parent center-of-mass frame. The children direction is obtained by uniformly sampling a unit vector on the 2-sphere (in the parent center-of-mass frame the children move in opposite directions). Finally, we apply a Lorentz boost to the lab frame, to obtain the 4 dimensional vector $p_\mu = (E, p_x, p_y, p_z)$ that characterizes each node. This prescription ensures *momentum conservation* and *permutation invariance*.

2.1 The generative process

The generative process depends on the following input parameters:

- p_0^μ: four-momentum of the jet (input value for the root node of the tree).
- t_0: initial squared mass.
- t_{cut}: cut-off squared mass to stop the showering process.
- λ: decaying rate for the exponential distribution.

Next, we describe the splitting of a node as follows:

1. Draw t_{L} and t_{R} from an exponential distribution as follows,

$$t_L \sim f(t|\lambda, t_{\mathrm{P}}) = \frac{1}{1-e^{-\lambda}} \frac{\lambda}{t_{\mathrm{P}}} e^{-\frac{\lambda}{t_{\mathrm{P}}} t} \tag{3}$$

$$t_R \sim f(t|\lambda, t_{\mathrm{P}}, t_L) = \frac{1}{1-e^{-\lambda}} \frac{\lambda}{(\sqrt{t_{\mathrm{P}}} - \sqrt{t_L})^2} e^{-\frac{\lambda}{(\sqrt{t_{\mathrm{P}}} - \sqrt{t_L})^2} t} \tag{4}$$

   and define $m_L = \sqrt{t_{\mathrm{L}}}$ and $m_R = \sqrt{t_{\mathrm{R}}}$. We apply a veto on sampled values where $t_L \geqslant t_{\mathrm{P}}$ and $t_R \geqslant (\sqrt{t_{\mathrm{P}}} - \sqrt{t_L})^2$. For inference, given two particles, we assign $t_L \to \max\{t_L, t_R\}$ and $t_R \to \min\{t_L, t_R\}$.
2. Compute a 2-body decay in the parent rest frame.
3. Apply a Lorentz boost to each of the children, with $\gamma = \frac{E_p}{\sqrt{t_P}}$ and $\gamma\beta = |\vec{p}_p|/\sqrt{t_P}$.
4. If t_{L} (t_R) is greater than t_{cut} repeat the process.

The algorithm is outlined in Algorithm 1. After running, the final binary tree for the jet is obtained.

Algorithm 1: Toy Parton Shower Generator

```
function NodeProcessing (p^μ_p, t_P, t_cut, λ, tree)
  Input : parent momentum p⃗_p, parent mass squared t_p, cut-off mass squared t_cut, rate
          for the exponential distribution λ, binary tree tree
    Add parent node to tree.
    if t_p > t_cut then
        Sample t_L and t_R from the decaying exponential distribution.
        Sample a unit vector from a uniform distribution over the 2-sphere.
        Compute the 2-body decay of the parent node in the parent rest frame.
        Apply a Lorentz boost to the lab frame to each child.
        NodeProcessing (p^μ_p, t_L, t_cut, λ, tree)
        NodeProcessing (p^μ_p, t_R, t_cut, λ, tree)
```

2.2 Reconstruction: The Likelihood for a Proposed Jet Clustering

In addition to the generative model described above, which is used for generating data with Monte Carlo, we also need to be able to assign a likelihood value to a proposed jet clustering. To do this we use the same general form for the jet's likelihood based on a product of likelihoods over each splitting as in Eq. 1. In order to evaluate this we need to first reconstruct the parent from the left and right children. Then we use the same equations described above (Eq. 3 and 4) for the splitting probabilities that are used in the generative model. The `Ginkgo` library provides functions to evaluate the joint likelihood $p(x, z|\lambda, t_{\mathrm{cut}})$ of any proposed hierarchical clustering of the observed final state particles.

2.3 Greedy and beam search algorithms

As described in Sec. 1.1.1, we can reframe the goal of jet clustering as finding the maximum likelihood estimate (MLE) for the latent structure of a jet, given a set of constituents (leaves). Different algorithms will return different tree-like hierarchical clusterings z_{shower}, and we can compare the performance of various algorithms. We study approximate solutions for bottom-up agglomerative clustering like the generalized-k_t algorithms (which are a class of greedy algorithms that locally maximize the likelihood at each step in the clustering process) and beam search (which maximize the likelihood of multiple steps before choosing what to cluster).

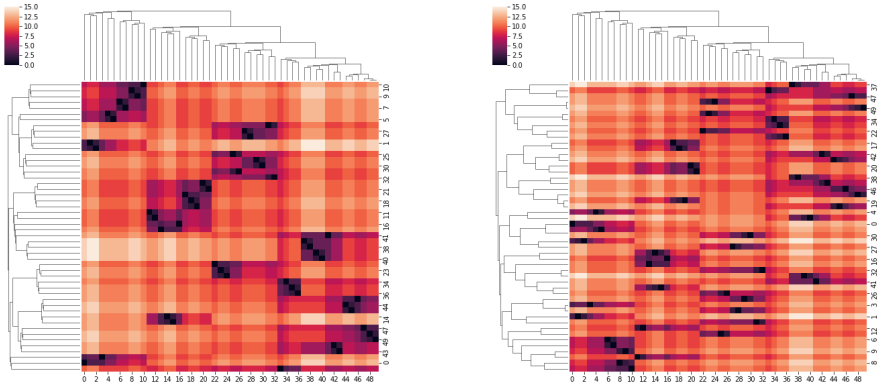

Figure 2. 2D heat clustermap visualizations where the leaves ordering corresponds to the order to access them when traversing the truth tree (columns) and each clustering algorithm (rows), where we show beam search (left) and k_t (right).

We provide implementations of these algorithms to jet physics in [25]. We also developed a visualization package in [26] and show examples below. In Fig. 2 we show 2D heat clustermaps where the color scale specifies the total number of steps needed to connect any two leaves through their closest common ancestor using the truth-level jet tree. The better the truth tree latent structure is reconstructed, the more the heat map structure looks block diagonal.

3 Examples of Research Enabled with Ginkgo

In this section we highlight some of recent research that has been enabled with `Ginkgo`.

3.1 Hierarchical Cluster Trellis Algorithm

Jet clustering in high-energy physics is a siloed sub-field of research, which is ironic given that hierarchical clustering is a common task in many areas of science and can be effectively abstracted. Hierarchical clustering is often used to discover meaningful structures, such as phylogenetic trees of organisms [27], taxonomies of concepts [28], and subtypes of cancer [29].

We define a hierarchical clustering as a recursive splitting of a dataset of elements, $X = \{x_i\}_{i=1}^{N}$ into subsets until reaching singletons, e.g. leaves of a binary tree. This can equivalently be viewed as starting with the set of singletons and repeatedly taking the union of sets until reaching the entire dataset.

The authors of Ref. [23] consider an energy-based probabilistic model for hierarchical clustering. The model is based on measuring the compatibility of each pair of sibling nodes, described by a potential function $\psi : 2^X \times 2^X \to \mathbb{R}^+$. We also denote the potential function for a hierarchical clustering $\mathtt{H}$ and dataset X as $\phi(X|\mathtt{H})$. Then, the probability of $\mathtt{H}$ for the dataset X, $P(\mathtt{H}|X)$, is equal to the unnormalized potential of $\mathtt{H}$ normalized by the partition function, $Z(X)$:

$$P(\mathtt{H}|X) = \frac{\phi(X|\mathtt{H})}{Z(X)} \quad \text{with} \quad \phi(X|\mathtt{H}) = \prod_{X_L, X_R \in \text{siblings}(\mathtt{H})} \psi(X_L, X_R) \tag{5}$$

where the partition function is given by $Z(X) = \sum_{\mathtt{H} \in \mathcal{H}(X)} \phi(X|\mathtt{H})$.

Next they define MAP hierarchy as the maximum likelihood hierarchical clustering given a dataset X. Typically, we want to exactly find the MAP hierarchy and the partition function.

However, a straightforward approach employing a brute force method to enumerate all hierarchical clusterings over N elements becomes infeasible because the number of hierarchies grows extremely rapidly, namely $(2N-3)!!$ [30, 31].

To overcome the computational burden, a cluster trellis data structure for hierarchical clustering was introduced in [23]. The trellis computes these quantities in the $O(3^N)$ time, without having to iterate over each possible hierarchy. While still exponential, this is feasible in regimes where enumerating all possible trees would be infeasible, and is to our knowledge the fastest exact MAP/partition function result, making practical *exact* inference for datasets on the order of 20 points (~3×10^9 operations vs ~10^{22} trees) or fewer.

We briefly review novel dynamic-programming algorithms for *exact* (and approx.) inference in hierarchical clustering introduced in [23]. The trellis allows us to compute the **partition function** $Z(X)$ and **MAP inference**, i.e. find the maximum likelihood tree structure. The Cluster Trellis package is available at https://github.com/SebastianMacaluso/ClusterTrellis. Each node in the trellis corresponds to all subsets of elements (jet constituents). A schematic representation and the assignment between nodes in a binary tree and nodes in the trellis is shown in Fig. 6.

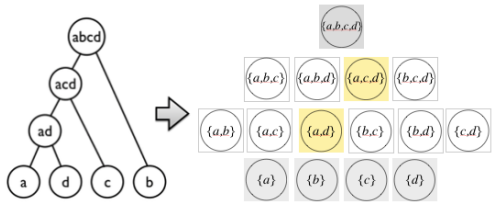

Figure 3. Schematic representation of the trellis and node assignment between the trellis and a binary tree.

Computing the Partition Function. Given a dataset of elements, $X = \{x_i\}_{i=1}^N$, the partition function, $Z(X)$, for the set of hierarchical clusterings over X, $\mathcal{H}(X)$, is given by Equation 3.1. The partition function for every node in the trellis is computed in order (in a bottom-up approach), memoizing the partial value at each node.

Computing the Maximum Likelihood Hierarchical Clustering. The MAP hierarchy for dataset X, $\mathtt{H}^\star(X)$, is $\mathtt{H}^\star(X) = \text{argmax}_{\mathtt{H}\in\mathcal{H}(X)} P(\mathtt{H}|X) = \text{argmax}_{\mathtt{H}\in\mathcal{H}(X)} \phi(\mathtt{H})$.

Sampling from the Posterior Distribution. Drawing samples from the true posterior distribution $P(\mathtt{H}|X)$ is also difficult because of the extremely large number of trees. However, there is a sampling procedure implemented using the trellis which gives samples from the exact true posterior without enumerating all possible hierarchies.

3.1.1 Sparse Cluster Trellis

The authors of Ref. [23] also introduced a sparse trellis data structure, which allows the algorithms to scale to larger datasets by controlling the sparsity index, i.e. the fraction of the total number of possible clusterings beingn considered. Most clusterings have likelihood values orders of magnitude smaller than the MAP clustering making their contribution to the partition function negligible. As a result, if we build a sparse trellis that considers the most relevant hierarchies, we could find approximate solutions for inference in datasets where implementing the full trellis is not feasible. The sparse trellis can be constructed from samples (e.g., ground truth from a simulator, greedy, or beam search trees) or randomly sample pairwise splittings for the children of a node.

3.1.2 Results

In Fig. 4 (left) we show the partition function versus the MAP hierarchy for each set of leaves from a Ginkgo dataset. Figure 4 (right) shows the results from sampling 10^5 hierarchies (black

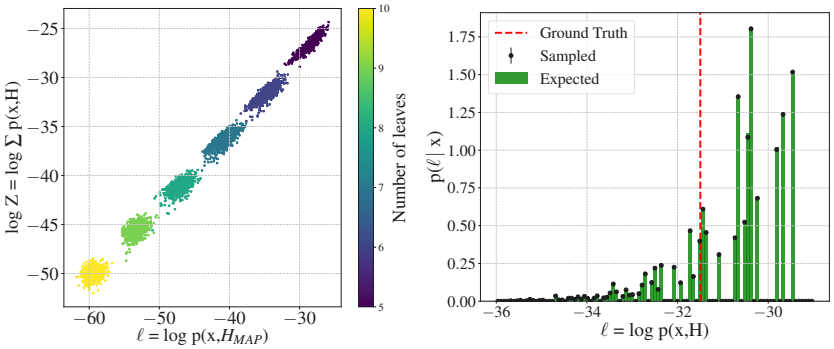

Figure 4. Left: scatter plot of the partition function Z vs. the maximum log likelihood for a Ginkgo dataset, with up to 10 jet constituents. The color indicates the number of leaves of each hierarchical clustering. Right: comparison of the posterior distribution for a specific jet with five leaves for sampling 10^5 hierarchies (black dots with small error bars) and expected posterior distribution (in green). The log likelihood for the ground truth tree is a vertical dashed red line.

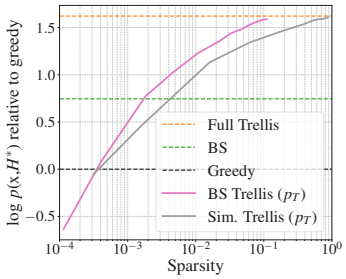

Figure 5. Trellises MAP hierarchy log likelihood (values are relative to the greedy algorithm) vs their sparsity. Each value corresponds to the mean over 100 trees of a test dataset. We show the Simulator (Sim.) and the Beam Search (BS) trellises.

dots) and the expected distribution Figure 5 shows the performance of the sparse trellis to calculate the MAP values on a set of 100 Ginkgo jets with 9 leaves. Even though beam search has a good performance for trees with a small number of leaves, we see that both sparse trellises quickly improve over beam search, with a sparsity index of only about 2%.

3.2 Hierarchical clustering through reinforcement learning

In this section we review results from [32] that cast hierarchical clustering as a Markov Decision Process (MDP) and adapted reinforcement learning algorithms to solve it. In particular, Monte-Carlo Tree Search (MCTS) guided by a neural network policy was adapted to the problem of jet clustering. This approach closely follows the AlphaZero algorithm [33–35], which achieved superhuman performance in a range of board games, demonstrating its ability to efficiently search large combinatorial spaces. While (model-free) RL methods have been used in the context of jet grooming, i.e. pruning an existing tree to remove certain backgrounds [36], they have not yet been used for clustering, that is, the construction of the binary tree itself.

3.2.1 Jet clustering as a Markov Decision Process

The authors of Ref. [32] used the ingredients of `Ginkgo` to recast the problem of clustering as an MDP, which is defined by the quartet $(\mathcal{S}, \mathcal{A}, P, R)$:

- The state space $\mathcal{S}$ is given by all possible particle sets at any given point during the clustering process, $s = z_t$.
- The actions $\mathcal{A}$ are the choice of two particles $a = (i, j)$ with $1 \leq i < j \leq n_t$ to be merged.
- The state transitions P are deterministic and update z_t to z_{t-1} by replacing the particles $p_{t,i}$ and $p_{t,j}$ with a parent $p_{t-1,i} = p_{t,i} + p_{t,j}$. All other particles are left unchanged, each state transition thus reduces the number of particles by one.

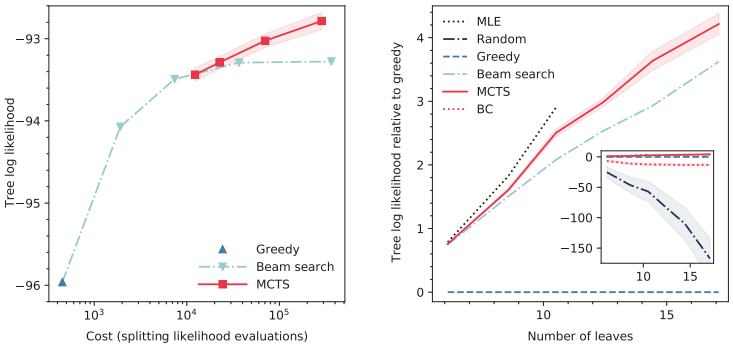

Figure 6. Mean log likelihood of clustered jets (larger is better). **Left**: against the computational cost, measured as the number of evaluations of the splitting likelihood p_s required by the different algorithms. **Right**: as a function of the number of final-state particles (leaves of the tree), using the best-performing (and most computationally expensive) hyperparameter setup for each algorithm. MCTS (solid, red) gives the highest-quality jet clusterings.

- The rewards R are the splitting probabilities, $R(s = z_t, a = (i, j)) = \log p_s(z_t|z_{t-1}(i, j))$.
- The MDP is episodic and terminates when only a single particle is left.

An agent solves the jet clustering problem by first considering the state of all observed, final-state particles and choosing which two to merge into a parent. It receives the log likelihood of this splitting as reward. Next, it considers the reduced set of particles where the two chosen particles have been replaced by their proposed parent, chooses the next pair of particles to merge, and so on. Rolling out an episode leads to a proposed clustering tree $z = \{z_1, \ldots, z_N\}$, with the total received reward being equal to the log likelihood of this tree following Eq. (1).

The formulation of jet clustering as an MDP allows us to use any (model-free) reinforcement learning (RL) algorithm to tackle it. Since the state transition model is known (and deterministic), they instead use a model-based planning approach to leverage this knowledge. They chose Monte Carlo Tree Search (MCTS) [33], which builds a search tree over possible clusterings z by rolling out a number of clusterings. In addition, the authors considered a clustering algorithm based on imitation learning, specifically Behavioral Cloning (BC): a policy π is trained to imitate the actions that reconstruct the true trees, which we can extract from the generative model, by maximizing $\log \pi(s, a_{\text{truth}})$.

3.2.2 Results

We present a comparison of the different clustering algorithms on a dataset of Ginkgo jets taken from [32]. They compare MCTS (with $c = 1$) and BC agents to a greedy algorithm that at each state picks the action with the maximum splitting likelihood p_s, a beam search algorithm that maintains the $b = 1000$ most likely clusterings while descending down the search tree, and a random policy. For jets with a small number of final-state particles we also compute the trellis exact maximum likelihood tree (MLE) following Ref. [23].

Fig. 6 shows the log likelihood of the clustering against the computational cost of the clustering algorithms (left) and against the number of final-state particles (right). While the greedy and beam search baselines lead to a robust performance at low computational cost, MCTS planning can generate hierarchical clusterings of a markedly higher likelihood. This advantage is more pronounced at larger number of final-state particles, showing that MCTS can explore large combinatorial spaces better than the baselines.

3.3 Probabilistic Programming

The Monte Carlo simulators implicitly describe the complicated distribution $p(x, z|\theta)$ and implement sampling through random number generators. Probabilistic programming extends

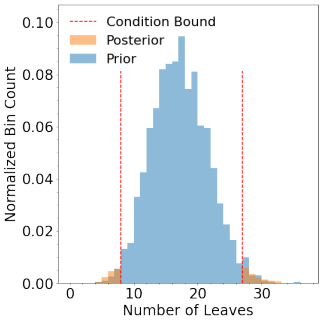

Figure 7. Histogram of the number of jet constituents (leaves) for jets generated with Ginkgo. We show the distribution of the number of constituents with no constraints (blue) and the one when using importance sampling with the limits on the number of leaves defined by the red dashed lines (orange).

this functionality with the ability to condition on the values of some of the random variables x or z [37], and it achieves this by hijacking the random number generators. This controlling inference algorithm uses those hooks to bias the simulator towards the desired output (e.g. importance sampling [38]) or through Monte Carlo sampling. In particular, it provides the ability to sample latent variables conditioned on observations, i.e. $z \sim p(z|x,\theta)$, and observations conditioned on latent variables, i.e. $x \sim p(x|z,\theta)$. For example, this technique can be used to efficiently sample the tails of backgrounds in signal-rich regions of phase space $z \sim p(z|\mathbf{1}(x),\theta)$.

As a proof of concept, we chose to use the `PyProb` framework [39] (applied to the `Sherpa` event generator in Refs. [40]) to implement probabilistic programming in Ginkgo. After integrating `PyProb` and Ginkgo, we successfully sampled jets while conditioning on variables, such as the jet transverse momentum and the number of constituents. The histogram in Fig. 3.3 demonstrates one simple example of the effectiveness of this framework for sampling tails of distributions. Using `PyProb`, we are able to force Ginkgo to only produce samples of jets with fewer than eight or more than twenty-six constituents. We can see that importance sampling accurately samples the desired regions of the distribution, i.e. with the same relative probabilities as the original distribution. Though this is a simple example, this method is powerful enough to allow us to condition on any sampled value within the generator, including latent variables, and condition on those values or arbitrary combinations of them.

4 Conclusion

This paper introduces a framing of common tasks in jet physics in probabilistic terms. We present `Ginkgo`, a simplified generative model for jets designed to facilitate research into new computational techniques for jet physics. A novel trellis data structure and dynamic programming algorithms that have been developed for hierarchical clustering motivated by this work. The `Ginkgo` library has been interfaced with both an implementation of the trellis algorithm and the Open AI Gym reinforcement learning library. We presented comparisons of jet clustering using greedy, beam search, Monte Carlo Tree Search, and the sparse and full cluster trellis. These new algorithms provide a principled alternative to the generalized k_t algorithms, which are based on a greedy sequential clustering algorithm. Additionally, we show that the trellis allows to marginalize and sample from the true posterior distribution of clustering histories for a set of jet constituents. This could ameliorate bottlenecks when implementing the CKKW-L matching algorithm for events with large jet multiplicity. Finally, we presented how complicated regions of phase space could be sampled using probabilistic programming.

Acknowledgements

We would like to thank Craig S. Greenberg, Nicholas Monath, Ji-Ah Lee, Patrick Flaherty, Andrew McGregor, and Andrew McCallum for their collaboration on efficient maximum

likelihood estimation on Ginkgo with a trellis. We are grateful for the support of the National Science Foundation under the awards ACI-1450310, OAC-1836650, and OAC-1841471, the Moore-Sloan data science environment at NYU, as well as the NYU IT High Performance Computing resources, services, and staff expertise.

References

[1] T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C.O. Rasmussen, P.Z. Skands, Comput. Phys. Commun. **191**, 159 (2015), 1410.3012

[2] J. Bellm et al., Eur. Phys. J. C **76**, 196 (2016), 1512.01178

[3] T. Gleisberg, S. Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert, J. Winter, JHEP **02**, 007 (2009), 0811.4622

[4] J. Brehmer, K. Cranmer, G. Louppe, J. Pavez, Phys. Rev. D **98**, 052004 (2018), 1805.00020

[5] K. Cranmer, J. Brehmer, G. Louppe, *The frontier of simulation-based inference* (National Academy of Sciences, 2020), ISSN 0027-8424, https://www.pnas.org/content/early/2020/05/28/1912789117.full.pdf

[6] M. Cacciari, G.P. Salam, G. Soyez, JHEP **04**, 063 (2008), 0802.1189

[7] S. Catani, Y.L. Dokshitzer, M.H. Seymour, B.R. Webber, Nucl. Phys. B **406**, 187 (1993)

[8] Y.L. Dokshitzer, G.D. Leder, S. Moretti, B.R. Webber, JHEP **08**, 1 (1997), 9707323

[9] S.D. Ellis, D.E. Soper, Phys. Rev. D **48**, 3160 (1993), 9305266

[10] A. Buckley, H. Hoeth, H. Lacker, H. Schulz, J.E. von Seggern, Eur. Phys. J. C **65**, 331 (2010), 0907.2973

[11] J. Brehmer, K. Cranmer, G. Louppe, J. Pavez, Phys. Rev. Lett. **121**, 111801 (2018), 1805.00013

[12] J. Brehmer, G. Louppe, J. Pavez, K. Cranmer (2018), 1805.12244

[13] A. Andreassen, B. Nachman, Phys. Rev. D **101**, 091901 (2020), 1907.08209

[14] G. Louppe, J. Hermans, K. Cranmer (2017), 1707.07113

[15] A.G. Baydin et al., *Etalumis: Bringing Probabilistic Programming to Scientific Simulators at Scale* (2019), 1907.03382

[16] A.G. Baydin et al. (2018), 1807.07706

[17] S. Catani, F. Krauss, R. Kuhn, B. Webber, JHEP **11**, 063 (2001), hep-ph/0109231

[18] L. Lonnblad, JHEP **05**, 046 (2002), hep-ph/0112284

[19] D.E. Soper, M. Spannowsky, Phys. Rev. D **84**, 074002 (2011), 1102.3480

[20] D.E. Soper, M. Spannowsky, Phys. Rev. D **87**, 054012 (2013), 1211.3140

[21] D. Ferreira de Lima, P. Petrov, D. Soper, M. Spannowsky, Phys. Rev. D **95**, 034001 (2017), 1607.06031

[22] S. Höche, S. Prestel, H. Schulz, Phys. Rev. D **100**, 014024 (2019), 1905.05120

[23] C.S. Greenberg, S. Macaluso, N. Monath, J.A. Lee, P. Flaherty, K. Cranmer, A. McGregor, A. McCallum (2020), 2002.11661

[24] K. Cranmer, S. Macaluso, D. Pappadopulo, *Toy Generative Model for Jets Package* (2019), https://github.com/SebastianMacaluso/ToyJetsShower

[25] Cranmer, Kyle and Macaluso, Sebastian and Pappadopulo, Duccio, *Greedy and Beam Search clustering algorithms for jet physics* (2019), https://github.com/SebastianMacaluso/StandardHC

[26] Cranmer, Kyle and Macaluso, Sebastian and Pappadopulo, Duccio, Visualize Binary Trees Package (2019), https://github.com/SebastianMacaluso/VisualizeBinaryTrees

[27] A. Kraskov, H. Stögbauer, R.G. Andrzejak, P. Grassberger, EPL (Europhysics Letters) **70**, 278 (2005)

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

[29] 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., Proceedings of the National Academy of Sciences **98**, 10869 (2001)

[30] D. Callan, *A combinatorial survey of identities for the double factorial* (2009), 0906.1317

[31] E. Dale, J. Moon, *The permuted analogues of three Catalan sets* (1993), 0378-3758

[32] J. Brehmer, S. Macaluso, D. Pappadopulo, K. Cranmer, *Hierarchical clustering in particle physics through reinforcement learning*, in *34th Conference on Neural Information Processing Systems* (2020), 2011.08191

[33] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., nature **529**, 484 (2016)

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., nature **550**, 354 (2017)

[35] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel et al., arXiv:1712.01815 (2017)

[36] S. Carrazza, F.A. Dreyer, Phys. Rev. **D100**, 014014 (2019), 1903.09644

[37] A.D. Gordon, T.A. Henzinger, A.V. Nori, S.K. Rajamani, in *Proceedings of the on Future of Software Engineering* (2014), pp. 167–181

[38] T.A. Le, A.G. Baydin, F. Wood, *Inference compilation and universal probabilistic programming*, in *Artificial Intelligence and Statistics* (2017), pp. 1338–1348

[39] A.G. Baydin, L. Shao, W. Bhimji, L. Heinrich, S. Naderiparizi, A. Munk, J. Liu, B. Gram-Hansen, G. Louppe, L. Meadows et al., *Efficient probabilistic inference in the quest for physics beyond the standard model*, in *Advances in neural information processing systems* (2019), pp. 5459–5472

[40] A.G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe et al., *Etalumis: Bringing probabilistic programming to scientific simulators at scale*, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), pp. 1–24