QUARKS, HIERARCHICAL CLUSTERING, AND COMBINATORIAL OPTIMIZATION

@KyleCranmer
New York University
Department of Physics
Center for Data Science
CILVR Lab
In recent years, **deep learning** has significantly improved the fields of computer vision, natural language processing and speech recognition. Beyond these traditional fields, deep learning has been expanded to quantum chemistry, **physics**, neuroscience, and more recently to **combinatorial optimization** (CO).

- DL is particularly attractive to address CO problems given its high flexibility, **approximate nature**, and self-learning paradigm.
- In other words, DL has the potential to learn universal high-quality algorithms and therefore could lead to a breakthrough in traditional CO, where algorithms are hand-crafted.
- On the other hand, **synergies between DL and CO algorithms** could lead to the possibility of taking **the best of the two domains** and deriving new algorithms, especially for applied problems.
Quarks, Gluons, and Jets
Evolution of the tree is latent

We only observe the leaves
Jet Clustering

Physicists would like to reconstruct the unobserved latent tree $z$ from observed $x$

- Hierarchical clustering can be seen as inverting the generative process
- Standard techniques use bottom-up / greedy / agglomerative clustering HAC
- Similarity measure is motivated by underlying physics: QCD, relativity, etc.
Optimal solutions for many downstream tasks would be ~trivial if an oracle could give us the correct / ground-truth tree, but

- Several trees could potentially lead to the same set of leaves, so we need to think probabilistically

Traditional approaches for downstream tasks (eg. classification, regression, etc.) interpret the HAC tree as an estimate for ground truth, and additional work is needed to characterize how properties of the clustering algorithm impact downstream performance
Deep Learning

Deep learning has been used successfully for solving downstream tasks without explicitly inferring the latent state.

- Rich research area for networks working on structured data like sets, trees, and graphs.
- Some hybrid deep learning algorithms exploit physics knowledge directly (e.g., include HAC clustering internally).

I've been interested in hybrid physics-DL models that treat latent state probabilistically.

---

Generative Model

In reality the best we can hope for is to invert the process probabilistically

- Forward model is a Markov process with a relatively simple probability model for each splitting parametrized by $\theta$

$$p(\text{tree} | \theta) = \prod_{\text{nodes}} p(\text{left, right} | \text{parent}, \theta)$$

Inference on $\theta$ would be new

Reframe jet physics in statistical terms

- Joint likelihood $p(x, z | \theta)$
- Maximum likelihood history: $\text{ArgMax}_z p(x, z | \theta)$
- Marginal likelihood $p(x | \theta) = \int dz \ p(x, z | \theta)$
- Maximum likelihood parameter: $\text{ArgMax}_\theta p(x | \theta)$
- Posterior distribution on histories: $p(z | x, \theta)$
- Posterior distribution on $\theta$: $p(\theta | x)$
**Generative Model**

**Ginkgo** is a simplified jet simulator written in pyro to facilitate research on exact/approximate optimization and marginalization over hierarchical clusterings (the latent tree) using

- Probabilistic Programming
- Differentiable Programming
- Dynamic Programming
- Variational Inference

Captures essential physics:

- Energy/Momentum conservation
- Dokshitzer–Gribov–Lipatov–Altarelli–Parisi—like evolution

---

**Algorithm 1: Toy Parton Shower Generator**

1. function NodeProcessing \((p^0_t, t, t_{cut}, \lambda, \text{tree})\)
2. Input: parent momentum \(p^0_t\), parent mass squared \(t\), cut-off mass squared \(t_{cut}\), rate for the exponential distribution \(\lambda\), binary tree \(\text{tree}\)
3. Add parent node to tree.
4. if \(t > t_{cut}\) then
5. Sample \(t_L\) and \(t_R\) from the decaying exponential distribution.
6. Sample a unit vector from a uniform distribution over the 2-sphere.
7. Compute the 2-body decay of the parent node in the parent rest frame.
8. Apply a Lorentz boost to the lab frame to each child.
9. NodeProcessing \((p^0_t, t_L, t_{cut}, \lambda, \text{tree})\)
10. NodeProcessing \((p^0_t, t_R, t_{cut}, \lambda, \text{tree})\)

\[
\begin{align*}
t_L &\sim f(t|\lambda, t) = \frac{1}{1 - e^{-\lambda t}} e^{-\lambda t/t} \\
t_R &\sim f(t|\lambda, t, t_L) = \frac{1}{1 - e^{-\lambda (\sqrt{t_P} - \sqrt{t_L})^2}} e^{-\lambda (\sqrt{t_P} - \sqrt{t_L})^2/t}
\end{align*}
\]
Going beyond agglomerative clustering

With this probabilistic view, interesting to revisit standard agglomerative clustering

- Similarity heuristic: 
  \[ d_{ii'}^\alpha = \min(p_{ii}^{2\alpha}, p_{ii'}^{2\alpha}) \frac{\Delta R_{ii'}^2}{R^2} \]

- Generative model for jets: 
  \[ p(\text{tree} | \theta) = \prod_{\text{nodes}} p(\text{left}, \text{right} | \text{parent}, \theta) \]

- If \( p_{\text{node}}(\text{left}, \text{right} | \text{parent}, \theta) \) is monotonic with heuristic, then agglomerative clustering is a greedy algorithm to find an approximate MAP estimate
  \[ \hat{z} = \arg\max_z p(z | x, \theta) \]

Immediately suggests we can do better with beam search or some other search algorithm
The immediate problem is that the search space (the number of binary trees on unordered leaves) is enormous!! \((2N - 3)!!\)

- Fortunately, when Sebastian Macaluso was at UMass we connected with Nicholas Monath, Craig Greenberg, and Andrew McCallum’s group

Number of clustering histories for \(N\) leaves grows as
\[
a(N) = (2N - 3)!!
\]

<table>
<thead>
<tr>
<th># of leaves</th>
<th>Approx. # of trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>10k</td>
</tr>
<tr>
<td>9</td>
<td>2M</td>
</tr>
<tr>
<td>11</td>
<td>600M</td>
</tr>
<tr>
<td>150</td>
<td>(10^{300})</td>
</tr>
</tbody>
</table>

\[(2n - 1)!! = 2^n \cdot \frac{\Gamma \left( \frac{1}{2} + n \right)}{\sqrt{\pi}}\]
The Trellis Data Structure
We consider cost / energy / likelihood functions that are products over siblings

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

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

This includes:

- the likelihood for tree-like Markov models like our physics use case,
- a cost function used in cancer genomics,
- the Dasgupta cost
The trellis

A trellis is constructed, where nodes correspond to all subsets of elements

- The full trellis is large \((2^{|X|})\), but the number of trees is super-exponentially larger than the number of nodes in the trellis
The trellis is an efficient data structure for storing memoized values of the the partition function $Z(V)$ and $\phi(V|H^*)$ for the MAP clustering $H^*$ for each of subset of elements $V$

- This admits a **dynamic programming** algorithm to efficiently compute the **exact** partition function and MAP over all possible clusterings.
The sparse trellis

The problem with the full trellis is that it still grows exponentially

- But there is a natural **sparse trellis** that admits an **approximate** algorithm for the MAP clustering and partition function

- One simply removes nodes from the trellis (subject to a consistency condition)

- One can also construct a sparse trellis from samples (e.g., ground truth from a simulator, greedy, or beam search) or randomly sample pairwise splittings

![Diagram of the sparse trellis](image)

**Figure 7:** 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. In both cases, we present the trellis obtained by ordering the leaves of the input trees in increasing norm of their momentum vector \(~p_r^2\) (see the probabilistic model description of section 4.2 for more details). We compare sparse trellises with other orderings of the leaves in Appendix §A.12. We add the values of the exact trellis, beam search and greedy algorithms. The BS trellis approaches the performance of the full one for a smaller sparsity index than the Sim. Trellis. Also, the sparse trellises are pre-built and then run on new datasets (test), which is why BS performs better than BS trellis sometimes.

**Finally, as a proof of concept, we show in Figure 7 the performance of the sparse trellis to calculate the MAP values on a set of 100 Ginkgo jets with 9 leaves. We chose a dataset of 9 elements to be able to easily compare the performance of the sparse and full trellises. However, the sparse trellis can be applied to larger datasets. Even though beam search has a good performance for trees with a small number of leaves, we see that both sparse trellises quickly improve over beam search, with a sparsity index of only about 2%. Both sparse trellises approach the performance of the exact one, but the BS trellis does it sooner.**

**4.3 Cancer Genomics**

**Background**

Hierarchical clustering is a common clustering approach for gene expression data\(^4\). However, standard hierarchical clustering uses a greedy agglomerative or divisive heuristic to build a tree. It is not uncommon to have a need for clustering a small number of samples in cancer genomics studies. An analysis of data available from [clinicaltrials.gov](https://clinicaltrials.gov) shows that the median sample size for 7,412 completed phase I clinical trials involving cancer is only 30.
Efficient Sampling
Combinatorial Marginalization
Variational Inference
The partition function values $Z(V)$ for each node in the trellis encode a distribution over all possible trees

- One can populate a trellis with samples and think of the resulting trellis as a histogram over the discrete space of possible binary trees
- Transversing the trellis from top to bottom mirrors the (Markov) generative model we have in physics
- A softmax over the partition functions for nodes at each level leads to categorical distribution for splitting at that level
- This leads to a natural, efficient, and exact sampling algorithm
The partition function values $Z(V)$ for each node in the trellis encode a distribution over all possible trees

- This leads to a parametrization of distributions over the discrete space of possible binary trees
- We can also use a neural network or some other model to reparametrize $NN: \phi \rightarrow Z(V)$
- We can fit this parametrized distribution to samples of trees with maximum likelihood

We can also construct something like Variational Auto Encoder to approximate the posterior distribution $p(z|x)$ with a neural network parametrized trellis: $NN: x, \phi \rightarrow Z(V)$

- Together with efficient sampling could lead to efficient variational inference over the structured latent space
A* search + trellis

Preliminary work by:
Craig Greenberg*
Sebastian Macaluso*
Nicholas Monath*
Avinava Dubey
Patrick Flaherty
Manzil Zaheer
Amr Ahmed
Kyle Cranmer
Andrew McCallum
Most recently, we combined the trellis with A* search to find an efficient algorithm to find the global MAP hierarchical clustering that can leverage a heuristic:

- Compactly encode states in the space of hierarchical clusterings (as paths from the root to the leaves of the trellis), and
- Compactly represent the search frontier (as nested priority queues).

Partial Hierarchical Clustering State

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

Min-Heap Over Child Pairs

Store a separate heap at each node.
Approximate A* search + trellis

We can limit the size of the priorities queues and/or number of trees explored

- Yields an approximate A* search
- We can control the computational complexity of the search
- This allows us to scale the algorithms to many more elements

Store a separate heap at each node.
The resulting algorithm allows us to extend the search for the exact MAP / lowest-cost hierarchical clustering to more elements (from $10^{12}$ to $10^{15}$ trees).

The approximate algorithm improves over baselines, even in enormous search spaces with 150 elements ($10^{300}$ trees).

- Also non-physics benchmark with 500 elements (more than $10^{1000}$ trees)
The approximate A* algorithm has controlled run time even for ~150 elements.

The number of trees explored by the approximate A* algorithm is tiny compared to the $3^N$ nodes in the trellis, which is super-exponentially smaller than the number of trees.

Only about $10^{-180}$ of the total number of trees are explored in this case.
Hierarchical Clustering as a Markov Decision Process + Reinforcement Learning
Hierarchical Clustering as an MDP

In a separate, but related thread of research, we framed the hierarchical clustering as a Markov Decision Process:

- Interfaced to Open AI Gym
- Used various Reinforcement Learning algorithms

**Markov Decision Process (MDP).** We treat the problem of clustering as an MDP $(S, A, P, R)$:

- The state space $S$ is given by all possible particle sets at any given point during the clustering process, $s = z_t$.
- The actions $A$ 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.
- 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.

Figure 1: Jet clustering as a Markov Decision Process. States $s$ (squares) represent (partial) clusterings of the original particles (small circles), the agent begins in the unclustered state (top square). Each action $a$ chooses a pair of particles in the current state (which may be either part of the original particle set or the result of a partial clustering) to be merged next. The reward $r$ is the log likelihood of the corresponding $1 \rightarrow 2$ splitting.
Reinforcement Learning

In particular, we compared the performance of

- Monte Carlo Tree Search (MCTS)
- Imitation learning / Behavioral Cloning (BC).

to traditional baselines:

- Greedy
- Beam Search

And the exact MLE/MAP found by using the trellis algorithm.

Figure 2: Mean log likelihood of clustered trees (larger is better). We show the mean and its standard error between five models trained with different random seeds. \textbf{Left}: against the computational cost, measured as the number of evaluations of the splitting likelihood $p_s$ required by the different algorithms. For beam search and MCTS we show four different hyperparameter settings. \textbf{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 tree clusterings.
Hierarchical clustering is a common task in particle physics and many other areas

• The probabilistic view of hierarchical clustering as inverting generative model to infer a structured latent space is interesting and connects mechanistic models, variational inference, deep learning, and combinatorial optimization

The trellis data structure + dynamic programs for MAP and partition function look like a more classical algorithm, but

• The trellis represents a parametrized distribution over hierarchies
• We can differentiate the MAP and partition function w.r.t. these parameters
• We can parametrize this distribution with some model, like a neural network, and have amortized posteriors with efficient sampling (great for a VAE over discrete structures)
• All of these could be useful for reinforcement learning or model-based planning