



Hackjet: Improved jet clustering with SIFT and Heap-based agglomerative clustering

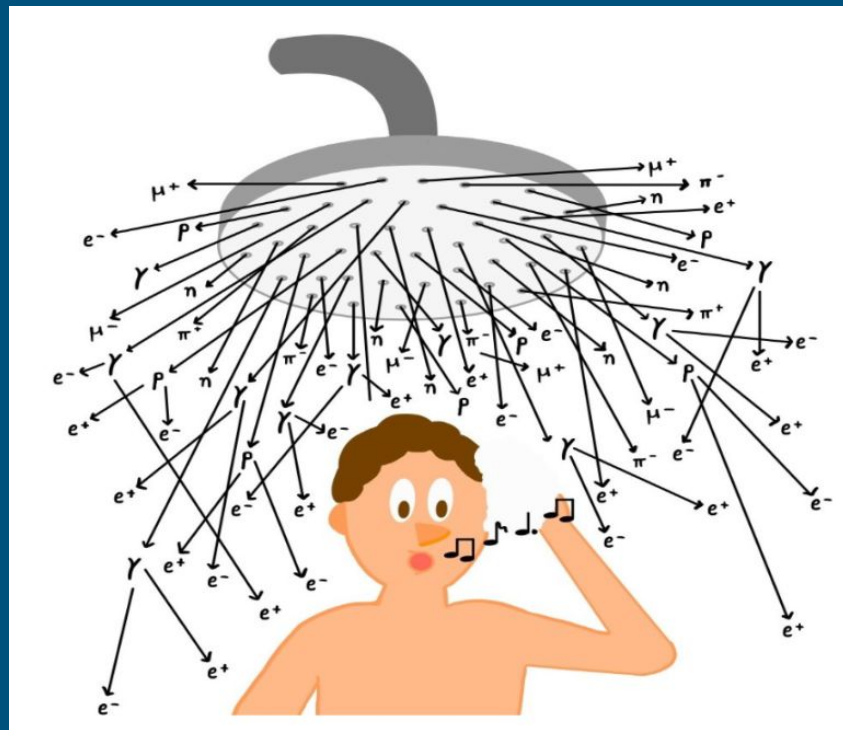


Alexei Tam
(Mentor: Kevin Pedro)
C2-THE-P2



Jet reconstruction

- As protons (and ions) collide in the CMS Detector, they pass through additional material that induces even more interactions!
- Result: showers and showers of additional particles, from the fragmentation and hadronization of quarks/gluons
- Investigation of these collimated showers (jets) $\Rightarrow$ Better understanding of properties and kinematics of their parent particles



There are lots of different tasks that go into reconstructing a jet:

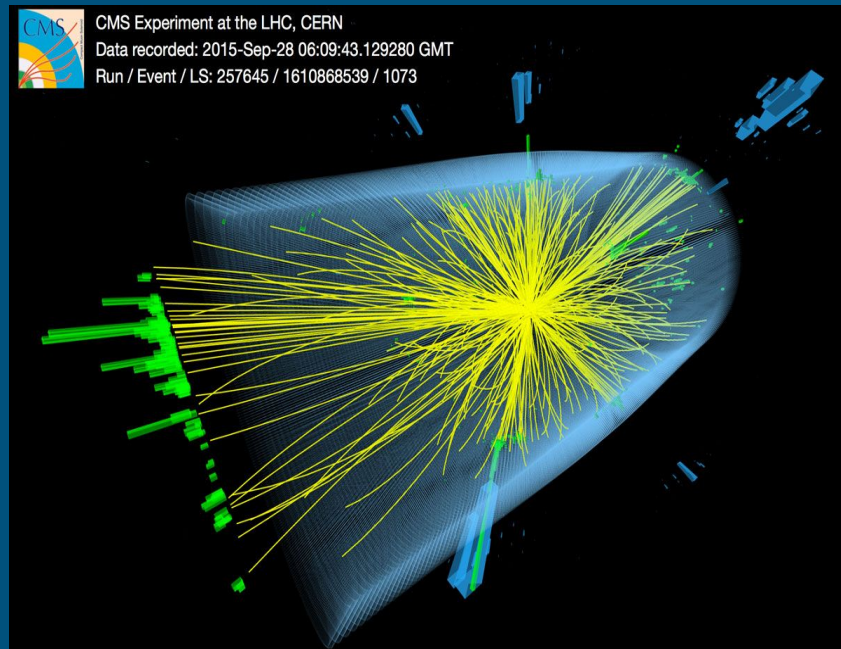
- Managing pileup
- Jet Tagging
- ParticleFlow and particle identification

Jet Clustering

But perhaps much more fundamental and interesting is the task of clustering itself!

Jets are (kind of) what you make them out to be; there are a variety of different ways to cluster them, and a variety of metrics by which one can measure the distance between clusters!

Metrics can reflect and account for the priorities going into the clustering (we'll get back to this in a moment)



Hold on cowboy...

- Is your metric:
 - Collinear Safe? [Can jet(s) change upon replacement of one particle w/ a collinear pair?]
 - Infrared Safe (?) [Can jet(s) change upon emission of a soft parton?]
- The kT-family of algorithms
- Can use n to adjust prioritization:
 - n = -1 (anti-kT; for clustering hard radiation first)
 - n = 0 (Cambridge-Aachen; purely based on geometric distance)
 - n = 1: (kT; for clustering soft radiation first)
 - R_0: jet size



$$d_{ij} = \min((p_T^i)^{2n}, (p_T^j)^{2n}) \frac{\Delta R_{ij}^2}{R_0^2}$$

$$\Delta R_{ij}^2 = \Delta \eta_{ij}^2 + \Delta \phi_{ij}^2$$

Scale-Invariant Filtered Tree (SIFT)

- A new jet clustering algorithm, with a cool distance metric!
- No need for a specified cone size (R_0)
 - May be really helpful for analyses in places like the dark sector, where one may be more in the dark (pun proudly intended) about how big the jet ought to be
- Still makes use of some of the useful behaviors of the kT-family
- Even comes with an additional metric, which determines whether:
 - I (Cluster): the two jets are clustered
 - II (Drop): the softer pair member is dropped
 - III (Isolate): the two pair members are isolated as final-state jets!

$$d_{ij} = \frac{\Delta m_{ij}^2}{(E_T^i)^2 + (E_T^j)^2} = \Delta \tilde{R}_{ij}^2 \star \epsilon_{ij}$$

$$\Delta \tilde{R}_{ij}^2 = \frac{\Delta m_{ij}^2}{E_T^i E_T^j}$$

$$\epsilon_{ij} = \frac{E_T^i E_T^j}{(E_T^i)^2 + (E_T^j)^2}$$

$$(I) : \delta_{ij} < (2\epsilon_{ij})^2$$

$$(II) : (2\epsilon_{ij})^2 \leq \delta_{ij} < 1$$

$$(III) : 1 \leq \delta_{ij}$$



- Dimensionless
- Lorentz Invariant
- Does not require R_0
- ~~Triangle Inequality~~

⇒ ~~FastJet~~

Jet Clustering as a Computational Problem

- Run 3 → HL-LHC going to only heighten the occupancy and computational challenges even further...
 - Higher collision energy (14 TeV)
 - $\sim 10^{34} \text{ cm}^{-2} \text{ s}^{-1}$ luminosity
 - 200 simultaneous collisions per bunch crossing
- Will eat up a lot of energy, effort, radiation tolerance...and time!
- N particles + iterating over each pair + comparing pairwise distances = $O(N^3)$

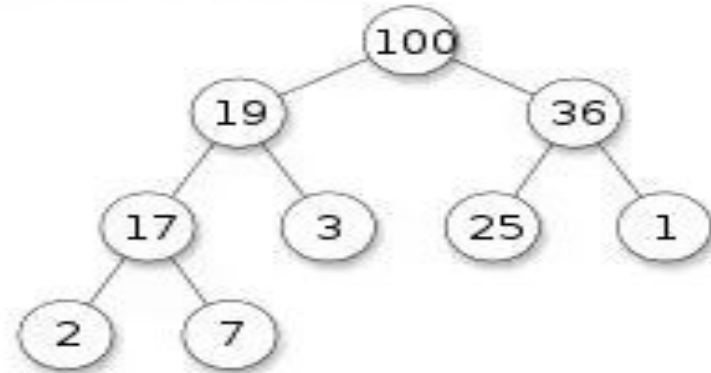


We've got all the figures together but I've no idea what order to put them in.

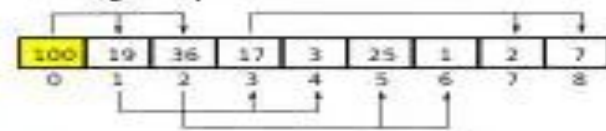
Heap-based Agglomerative Clustering

- Use a heap!
 - $H[i] \leq H[2i + 1]$ (for all $i = 1, \dots, N/2$), with N being the number of elements
- With this property, we know that the root node $H[1]$ is the least element
- With careful updating of the heap as we go along (removing elements, “shifting-up” and “shifting-down” elements), can cut down on complexity to $O(N^2 \log(N))$

Tree representation



Array representation



Build-a-Workflow

01

A “Naive” Baseline

02

FastJet, AEACuS

03

Comparison, Timing

04

Result, Evaluation

- “Naive” = an implementation with no speed-up mechanisms; gives us the baseline of how long the process we’re trying to speed up takes
 - kT-family and SIFT
- FastJet and AEACuS treated as “official” implementations (in the sense these are where our current optimization methods are at)
 - FastJet (kT-family)
 - AEACuS (SIFT)
- Comparison: need to make sure all compared implementations **only** vary in time-length; not varying in clustering itself
- Timing: to assess quality of optimization techniques being tested

The Naive Implementation

(“naive_outline.txt” → “NaiveAntikT.ipynb”, etc. → “naive.py”)



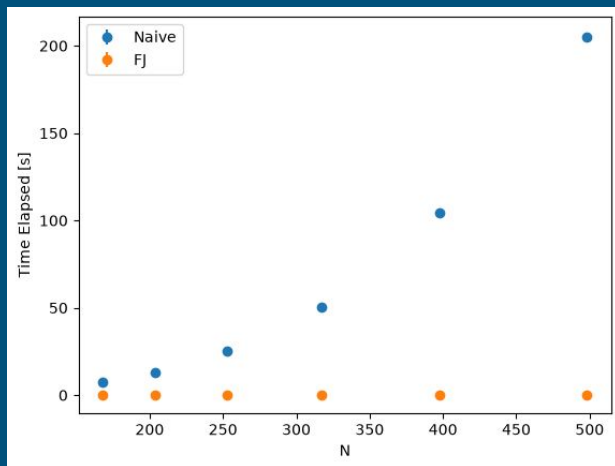
- Ability to create protojets (“MJets”) from input 4-momenta
- I/O: “reading” input .root files for 4-momenta to return arrays of protojets, and “writing” output .root files with 4-momenta of final-state jets
- No speed-ups; going through the whole nine yards previously described (calculating the distance matrix, making comparisons between pairs, etc.)
- GhostJetPopulator: a way to populate an input MJet set with ghost jets
- Also very slow...

FastJet and AEACuS

- FastJet
 - Has implementations for the entire kT-family (anti-kT, kT, Cambridge-Aachen)
- Very fast
- Relatively straightforward
- Plays relatively nicely with Naive implementation
 - PseudoJet ~ MJet
 - Similar I/O
 - etc.
- AEACuS (Algorithmic Event Arbiter and Cut Selector)
- By one of the authors of the SIFT paper, and unsurprisingly has a SIFT implementation we can compare against
- “a whole different kettle of fish”
- Python + Perl
- IN:
 - Events in a .lhco file
 - Instructions in a .dat file
- OUT:
 - Outputs in a .cut file
- Also a JetFilm notebook, to produce a video of the clustering process!

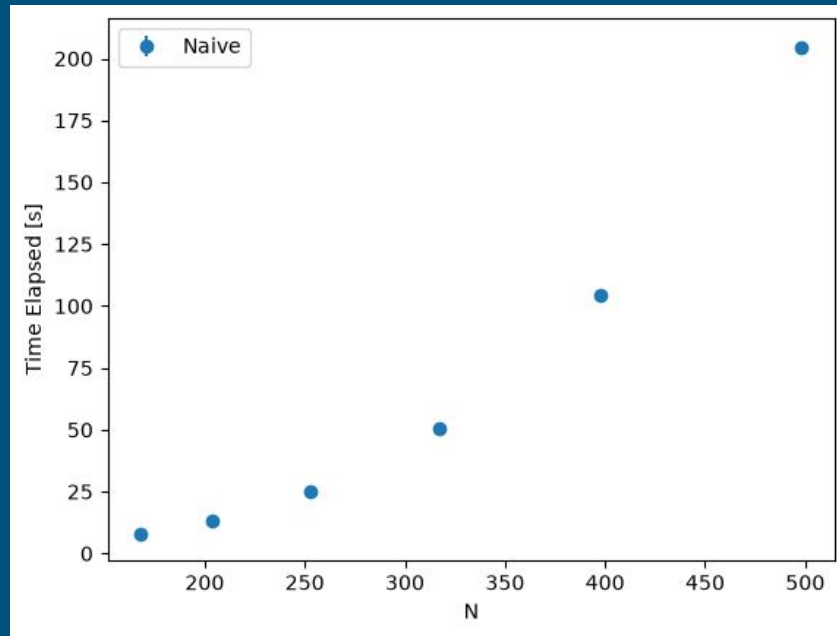
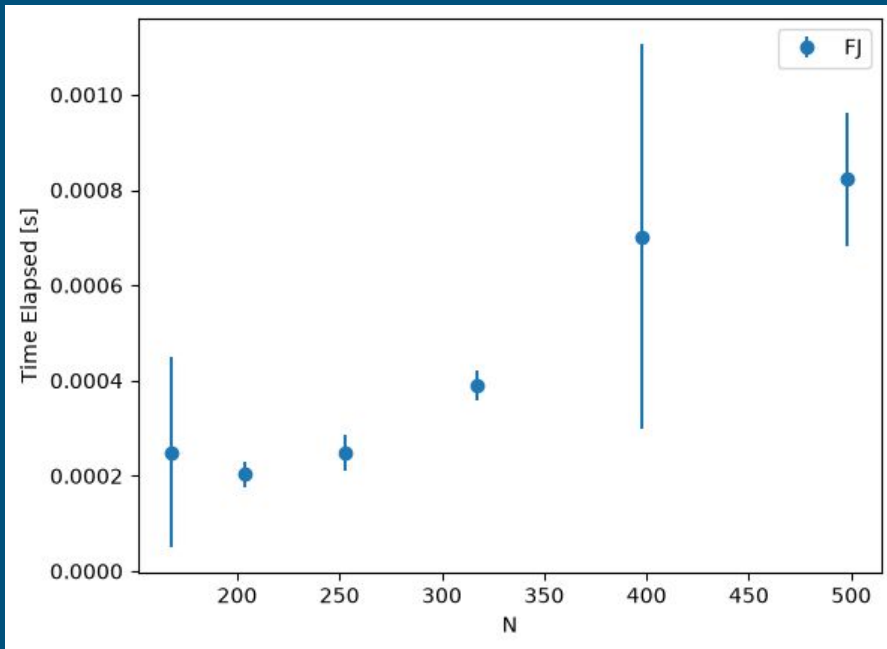
Comparison and Testing

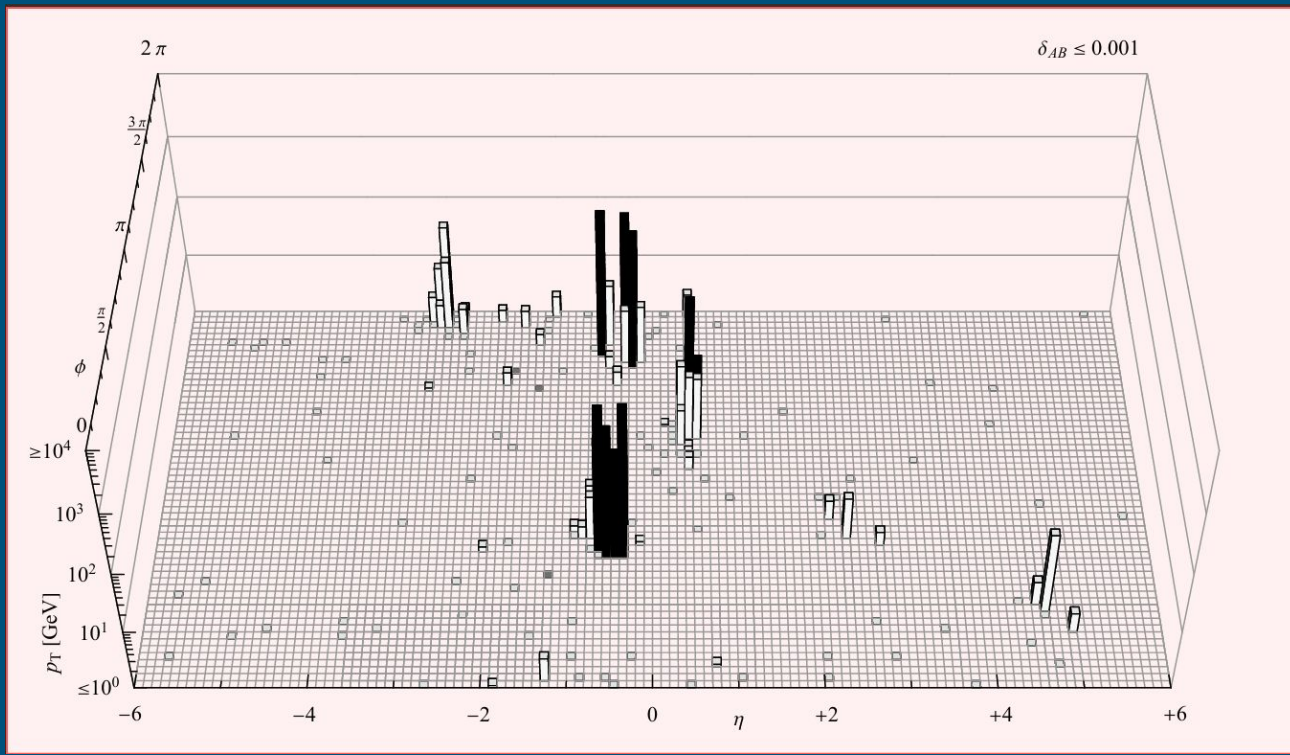
- Different ways to compare; this one
 - tags each input particle with an index
 - Output jets contain a list of indices, of the original input particles it contains
- Naive kT-family implementations ☐



```
Match found: FJ jet: [136] ; Naive jet: [136]
Match found: FJ jet: [95] ; Naive jet: [95]
Match found: FJ jet: [142] ; Naive jet: [142]
Match found: FJ jet: [92] ; Naive jet: [92]
Match found: FJ jet: [125] ; Naive jet: [125]
Match found: FJ jet: [101] ; Naive jet: [101]
Match found: FJ jet: [82] ; Naive jet: [82]
Match found: FJ jet: [127] ; Naive jet: [127]
Match found: FJ jet: [114] ; Naive jet: [114]
Match found: FJ jet: [7] ; Naive jet: [7]
Match found: FJ jet: [65, 118] ; Naive jet: [118, 65]
Match found: FJ jet: [98] ; Naive jet: [98]
Match found: FJ jet: [139] ; Naive jet: [139]
Match found: FJ jet: [91] ; Naive jet: [91]
Match found: FJ jet: [49] ; Naive jet: [49]
Match found: FJ jet: [107] ; Naive jet: [107]
Match found: FJ jet: [112] ; Naive jet: [112]
Match found: FJ jet: [108] ; Naive jet: [108]
Match found: FJ jet: [81] ; Naive jet: [81]
Match found: FJ jet: [105] ; Naive jet: [105]
Match found: FJ jet: [84] ; Naive jet: [84]
Match found: FJ jet: [109] ; Naive jet: [109]
Match found: FJ jet: [39] ; Naive jet: [39]
Match found: FJ jet: [43] ; Naive jet: [43]
Match found: FJ jet: [124] ; Naive jet: [124]
Match found: FJ jet: [15] ; Naive jet: [15]
```

More Timing Plots





- Successful calling of the perl script (aeacus.pl);
- Minimal test files (test .lhco file with 1-2 input particles) $\Rightarrow$ minimal results, but it works!

A frame of the clustering process (from JetFilm.nb in AEACuS)

Outlook

- Completing the validated workflow
 - Validating naive SIFT implementation against that of AEACuS
- C++ versions!
- Testing with the Heap-based method, to speed up SIFT to $O(N^2 \log N)$
- Implementing on GPUs