Jets in heavy ion collisions with fast clustering jet finders

Gavin Salam
work in progress with M. Cacciari

LPTHE, Universities of Paris VI and VII and CNRS

Quark Matter, Shanghai
November 2006

Jets in HI?

At RHIC, 'jet' studies look at high p_t particles and their average correlations.

Traditional (particle physics) jet studies instead seek to identify jets on an event-by-event basis, as reliable proxies for the 'original hard partons'.

To what extent (and how) can the traditional techniques be applied in the heavy-ion environment?

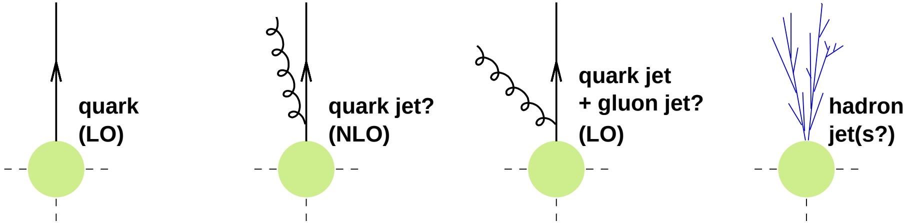
Talk has two parts

- Introduction to jet definitions.
- Overview of some progress relevant to HI.

Discussion will be in context LHC (where jets will be common)

Partons v. Jets?

Partons (quarks, gluons) are not trouble-free concepts...

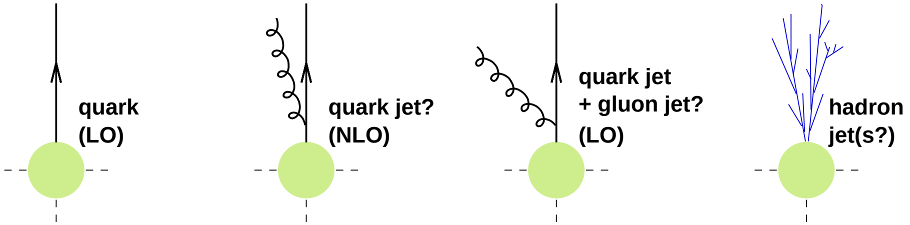


- Partons split into further partons
- Jets are a a way of thinking of the 'original parton'
- A 'jet' is a fundamentally ambiguous concept (e.g. requires a resolution)

Jets are only meaningful once you've defined a **jet algorithm**

Partons v. Jets?

Partons (quarks, gluons) are not trouble-free concepts...



- Partons split into further partons
- Jets are a a way of thinking of the 'original parton'
- A 'jet' is a fundamentally ambiguous concept (e.g. requires a resolution)

Jets are only meaningful once you've defined a **jet algorithm**

Guidelines for jet definitions

General

- Infrared and collinear safety – i.e. soft emissions and collinear splittings should not change jets
  otherwise pert. QCD cannot be used
- Definitions should be simple and detector independent
  otherwise different experiments cannot compare results

Specific to HI

- It must be computationally feasible to run on the $10^4 - 10^5$ particles expected at LHC.
- Procedure to reduce large background noise should also satisfy above 'safety' properties.

Jet finder defs.

Clustering jet finders

1. Calculate 'distances'
   - d_{ij} between all particles i and j
   - d_{iB} between i and beam
2. Find smallest of d_{ij} and d_{iB}
   - If d_{ij} is smallest, recombine i and j
   - if d_{iB} is smallest call i a jet
3. Goto step 1 if anything's left

Two variants (& one parameter, R)

- **k_t jet finder** [1991]

  $d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2, \quad d_{iB} = k_{ti}^2 R^2$

- **Cambridge/Aachen** [1998]

  $d_{ij} = \Delta R_{ij}^2, \quad d_{iB} = R^2 \qquad [\Delta R_{ij}^2 = \Delta y_{ij}^2 + \Delta\phi_{ij}^2]$

Jet finder defs.

Clustering jet finders

1. Calculate ‘distances’
   - d_{ij} between all particles i and j
   - d_{iB} between i and beam
2. Find smallest of d_{ij} and d_{iB}
   - If d_{ij} is smallest, recombine i and j
   - if d_{iB} is smallest call i a jet
3. Goto step 1 if anything’s left

Two variants (& one parameter, R)

- **$\mathbf{k_t}$ jet finder** [1991]

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2, \quad d_{iB} = k_{ti}^2 R^2$$

- **Cambridge/Aachen** [1998]

$$d_{ij} = \Delta R_{ij}^2, \quad d_{iB} = R^2 \qquad [\Delta R_{ij}^2 = \Delta y_{ij}^2 + \Delta\phi_{ij}^2]$$

Jet finder defs.

Clustering jet finders

1. Calculate 'distances'
   - d_{ij} between all particles i and j
   - d_{iB} between i and beam
2. Find smallest of d_{ij} and d_{iB}
   - If d_{ij} is smallest, recombine i and j
   - if d_{iB} is smallest call i a jet
3. Goto step 1 if anything's left

Two variants (& one parameter, R)

- **k_t jet finder** [1991]

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2, \quad d_{iB} = k_{ti}^2 R^2$$

- **Cambridge/Aachen** [1998]

$$d_{ij} = \Delta R_{ij}^2, \quad d_{iB} = R^2 \qquad [\Delta R_{ij}^2 = \Delta y_{ij}^2 + \Delta\phi_{ij}^2]$$

Cone jet finders e.g.

1. Create a seed (3-vector) from the direction of each input particles (possibly implement a way to specify a smaller list of seeds to save processing time ie. calo clusters).
2. For each seed, s, create a cone in η-ϕ space of radius R (set by the parameter radius around the seed axis such that a particle, p, with

$$(\eta_s - \eta_p)^2 + (\phi_s - \phi_p)^2 < R^2 \qquad (1)$$

   is defined to be inside the cone.
3. Then combine every particle in this cone into a jet using a $p_\perp$ recombination scheme as described in section 2.5.2 of the KtJet paper.
4. Now create a new cone around this jet's axis and repeat step 3. If the new jet's axis is colinear with the previous axis then the jet is stable and is added to the list of meta-jets, otherwise the process is repeated until either a stable jet if found or a maximum number of iterations is reached.
5. The next stage is, to enforce infra-red safety, to repeat steps 2-4 with a new set of seeds in-between every pair of jets i, j, found above if i and j are between 1 and 2 cone radii apart ie.

   if:

$$R^2 < (\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2 < (2R)^2 \qquad (2)$$

   then:

$$\eta_s = \frac{\eta_i + \eta_j}{2} \qquad \phi_s = \frac{\phi_i + \phi_j}{2} \qquad (3)$$

6. Next any jets with $p_\perp$ less than a pre-defined parameter epslon (typically of order 5 GeV) are removed from the list.
7. Then for each jet in the list, if the sum of the $p_\perp$s of any particles in the jet which are shared with a higher $p_\perp$ jet is greater than some fraction, ovlim, of this jet's $p_\perp$, then remove the jet from the list.
8. Next for each particle that is still in more than one jet, remove the particle from all but the closest jet to particle's direction, ie. the jet with the smallest $\Delta(\eta)^2 - \Delta(\phi)^2$.
9. Finally step 6 is repeated.

[from W. Plano]

Concrete example

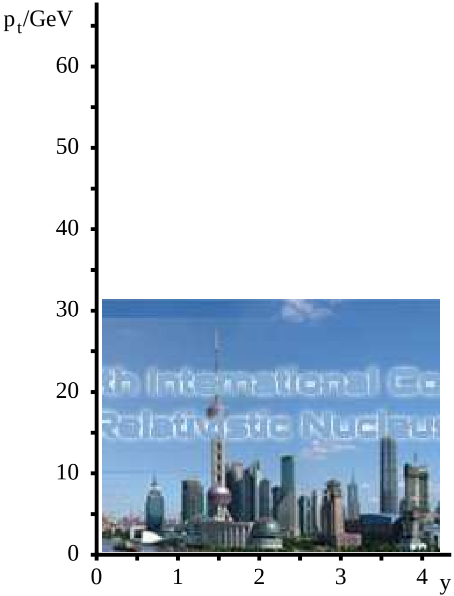
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



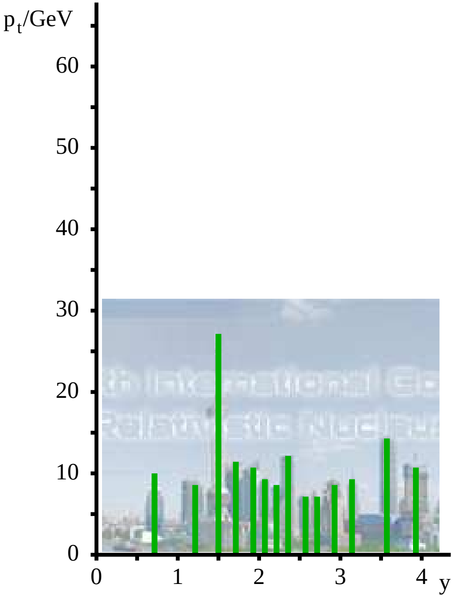
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



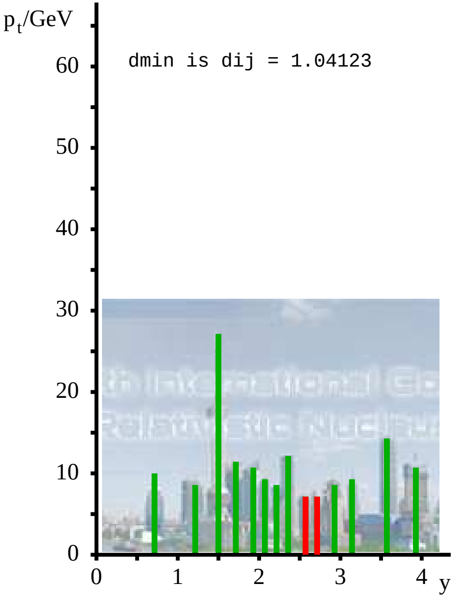
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



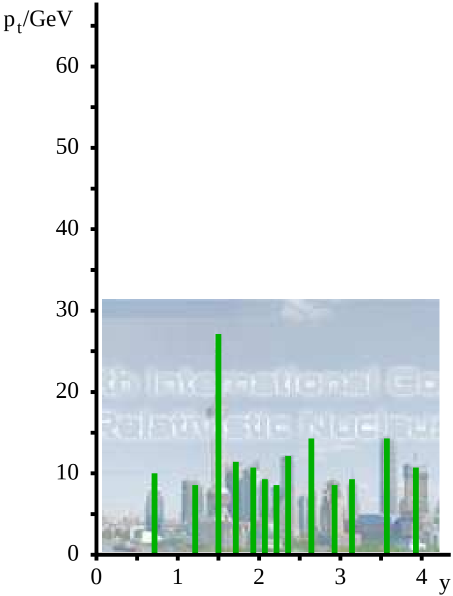
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



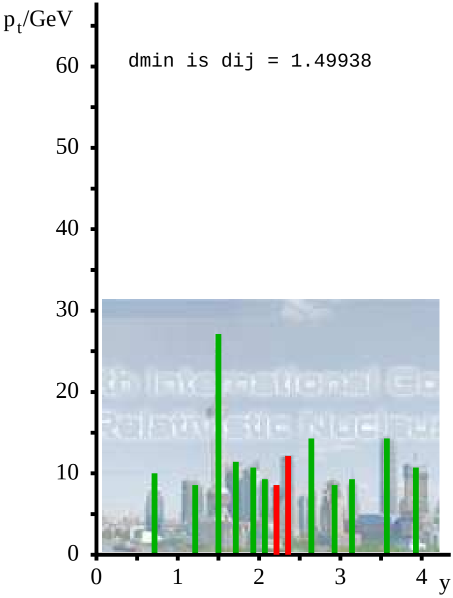
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



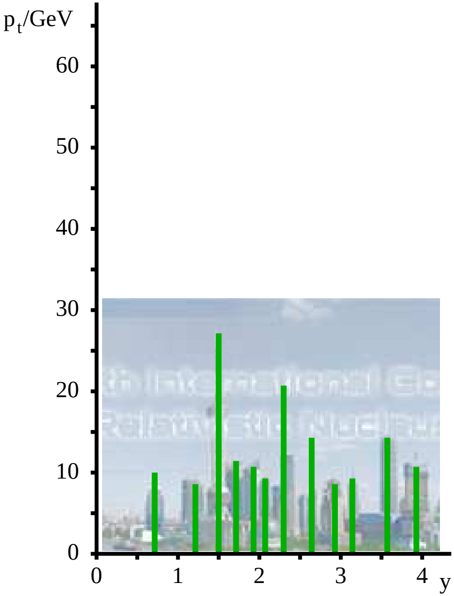
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



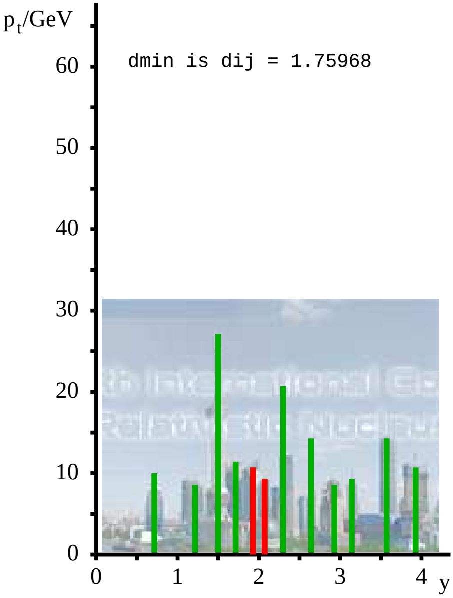
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



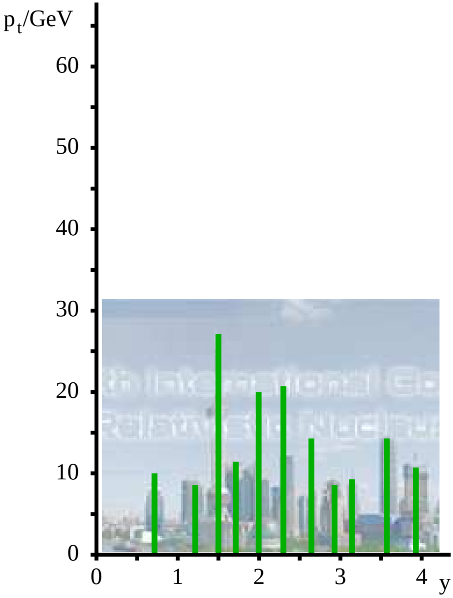
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



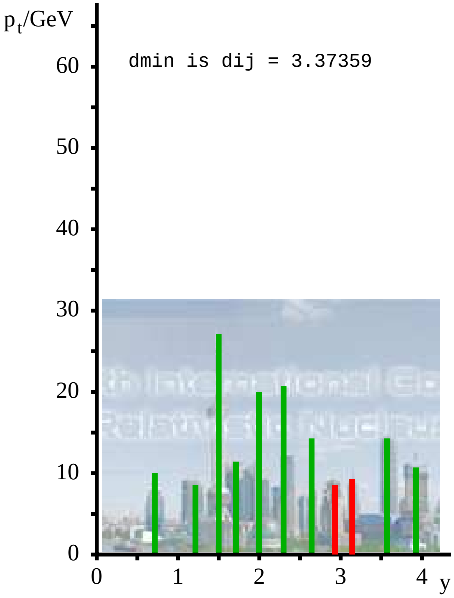
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



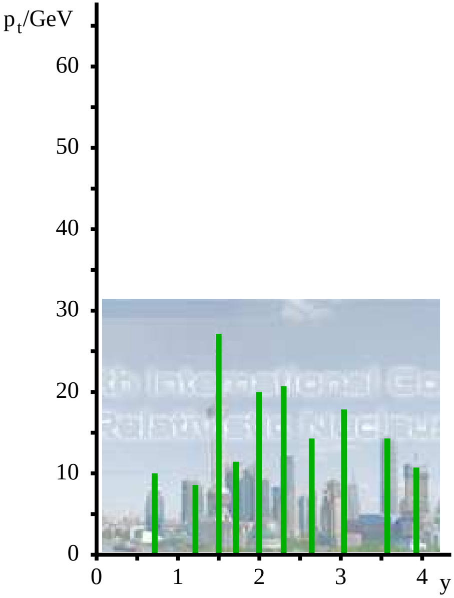
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



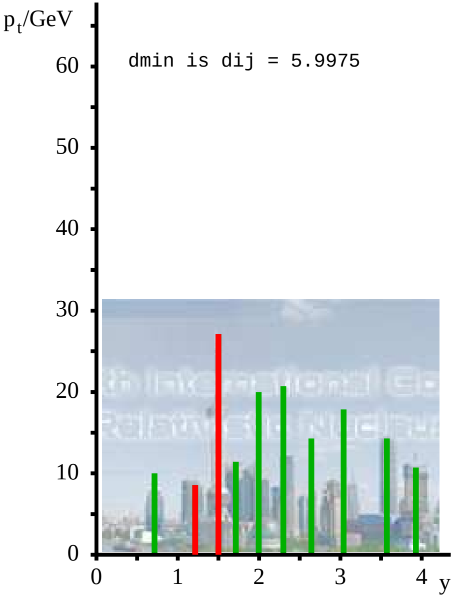
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



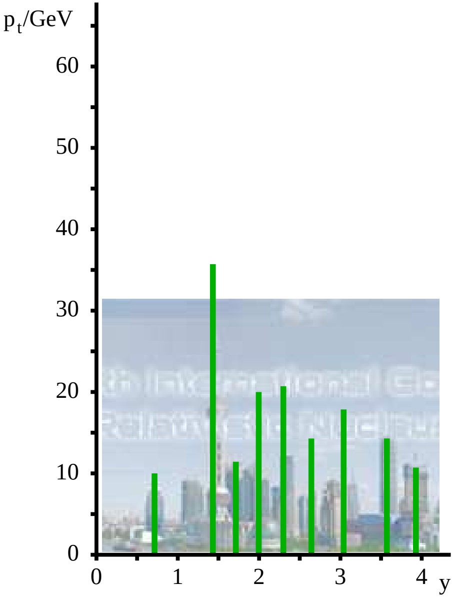
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



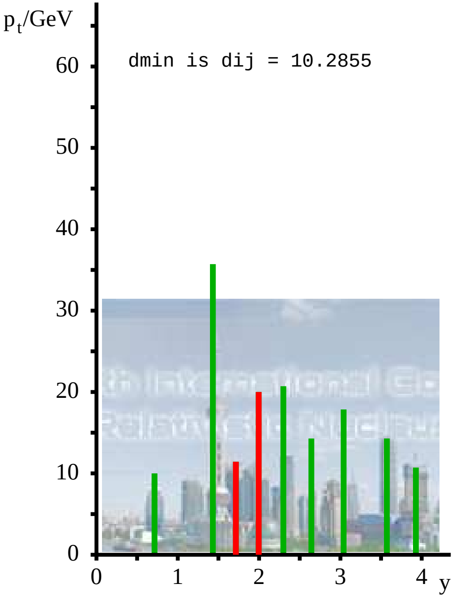
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



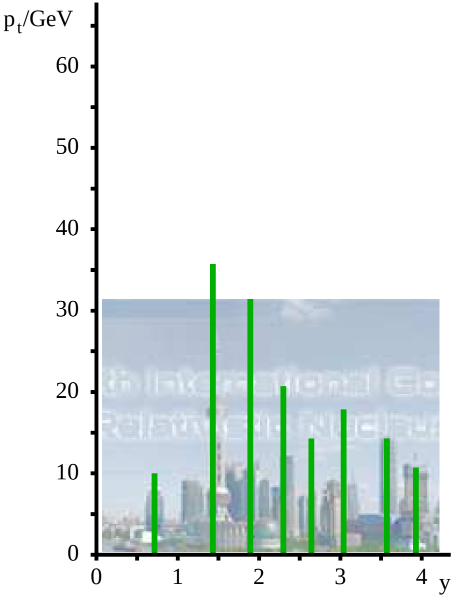
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



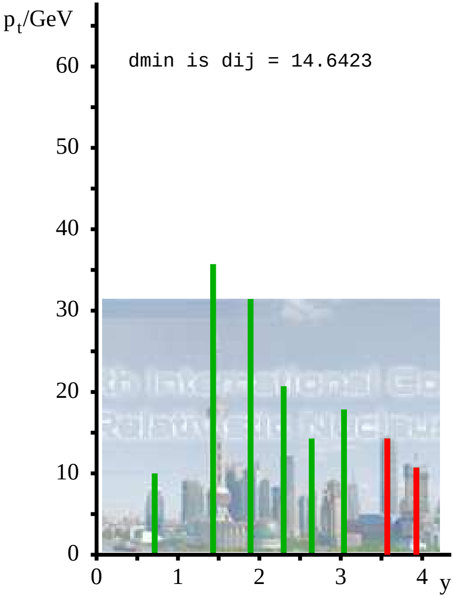
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



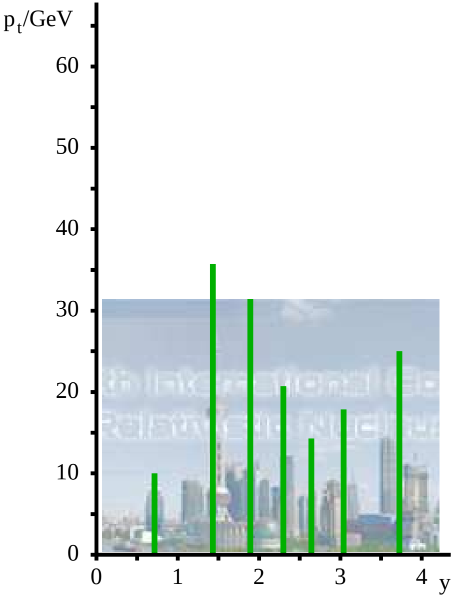
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



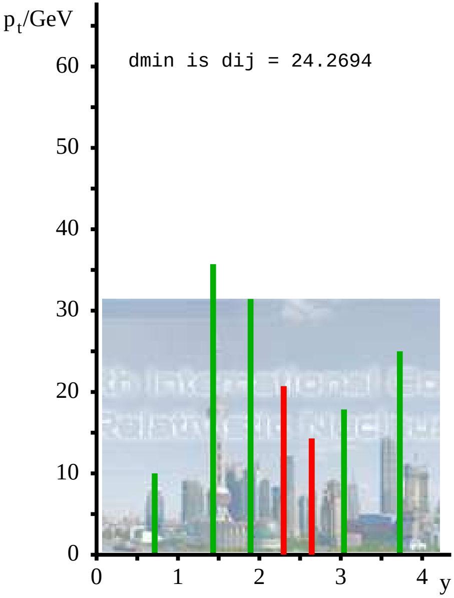
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



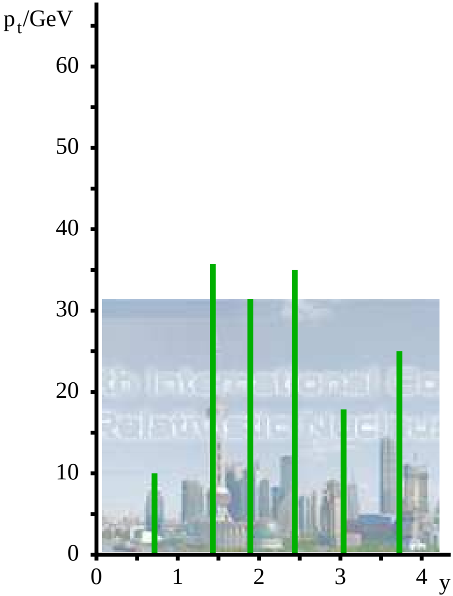
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



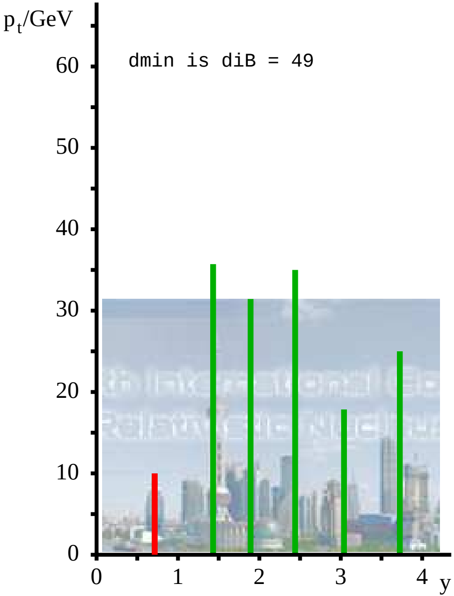
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



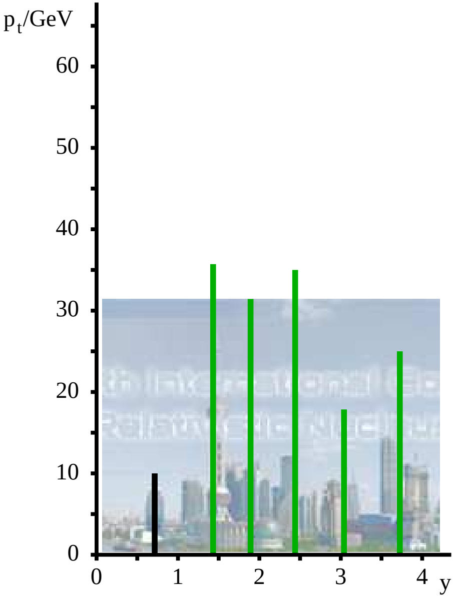
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



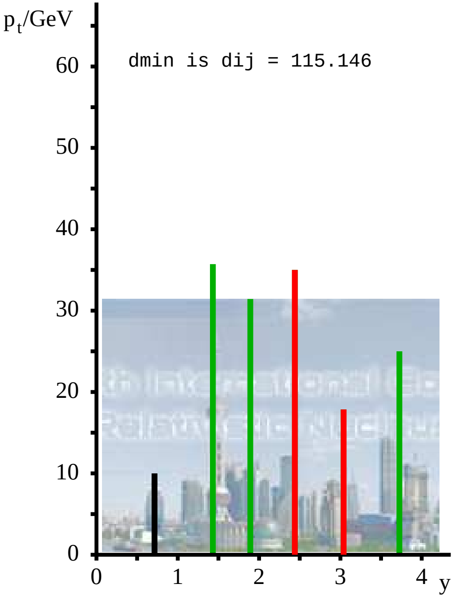
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



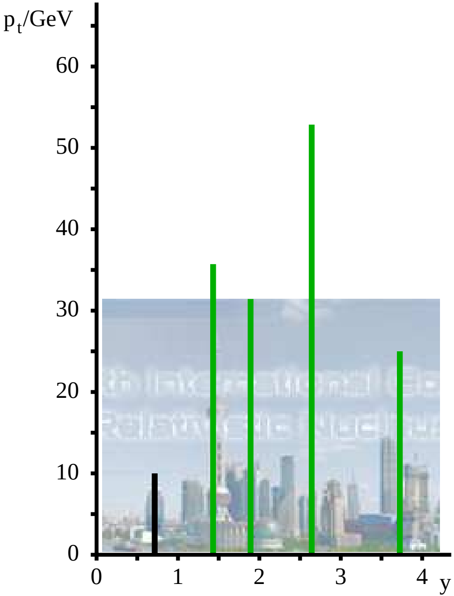
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



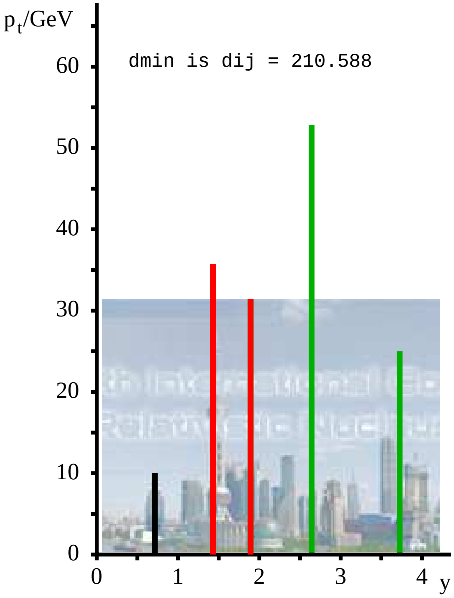
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



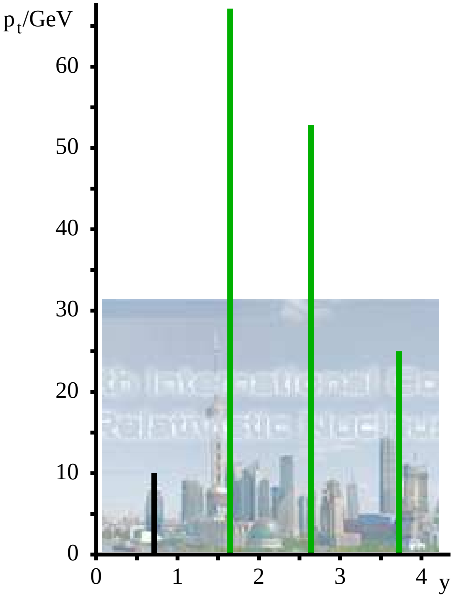
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



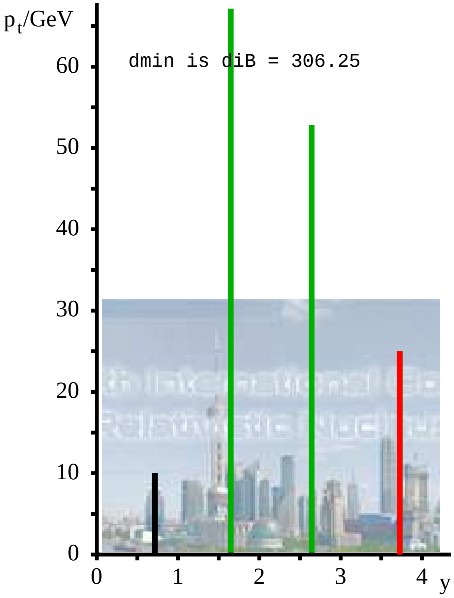
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary...

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



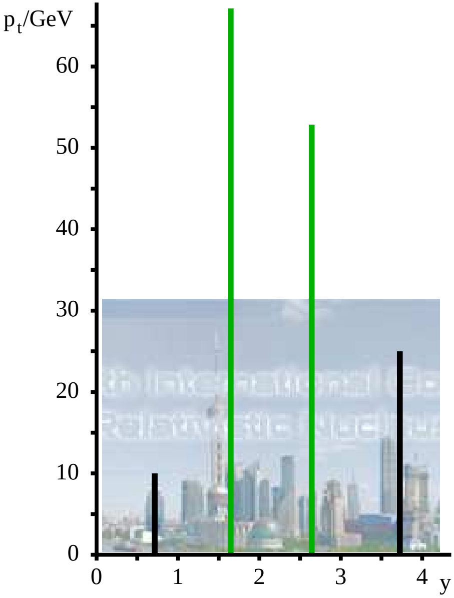
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



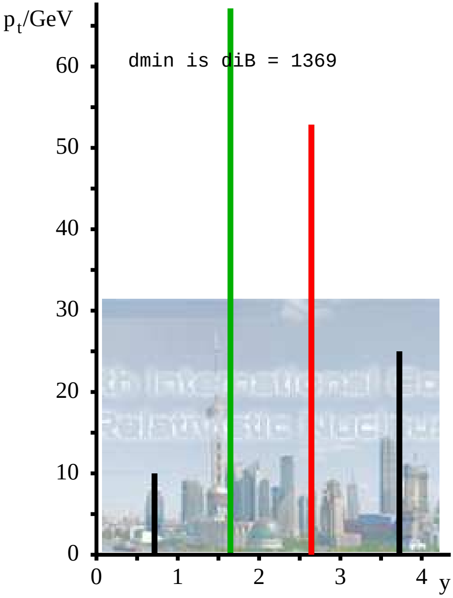
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary…

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



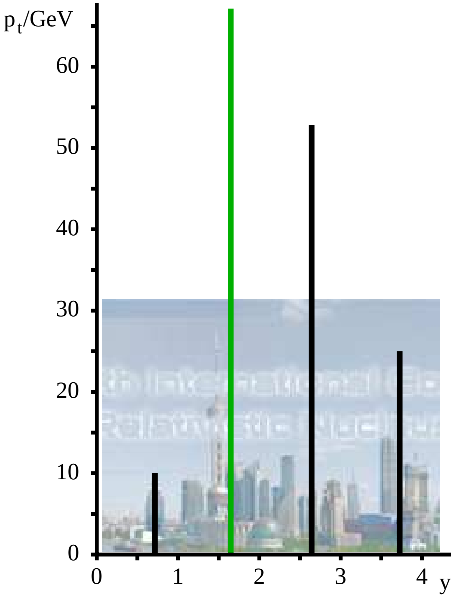
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



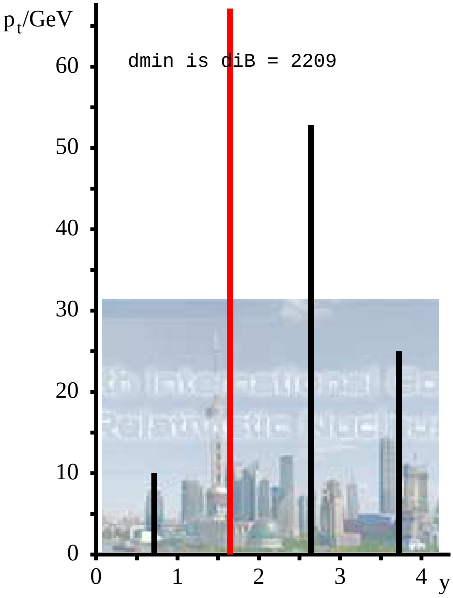
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



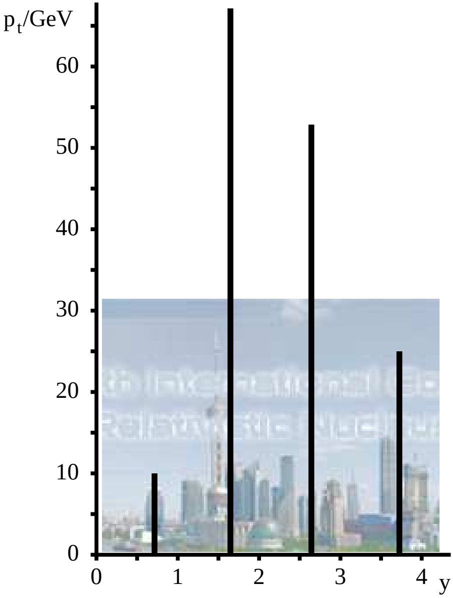
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



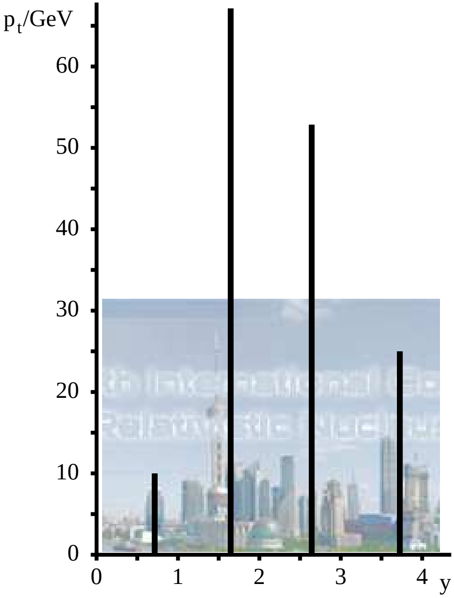
Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Concrete example



Example clustering with k_t algorithm, $R = 0.7$

ϕ assumed 0 for all towers

For Shanghai skyline, clustering is a bit arbitrary. . .

But on QCD events, *d_{ij} is related to divergences for branching* — clustering attempts inverse branching.

Advantages of clustering jet finders

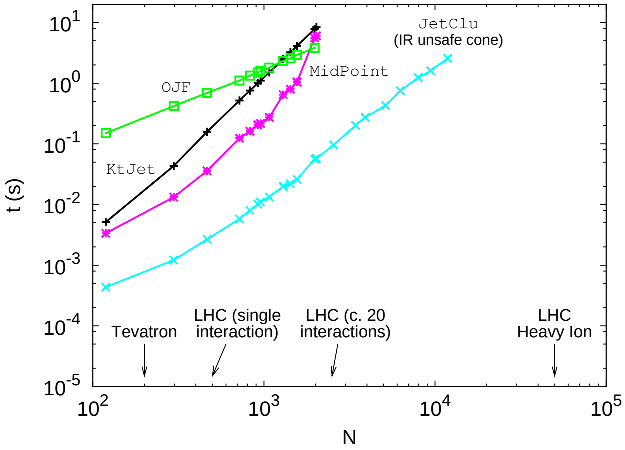
- They are always infrared & collinear safe.
- Simplicity $\rightarrow$ extensively studied theoretically
- They have smaller hadronization corrections than cone jet finders.

  More robust wrt fine details of quenching?
- They are the standard in e^+e^- and DIS colliders
- Starting to be used at Tevatron

Issues for HI:

- Long believed to be too computationally complex for high-multiplicity environments ($N > 1000$).
- Subtraction of "background" had never been attempted

Time to cluster N particles



Standard C++ (and fortran) k_t-clustering takes time $\sim N^3$.

a Pb-Pb event takes 1 day!

JetClu (cone) *is fast*. But IR unsafe.

Discontinued at Tevatron

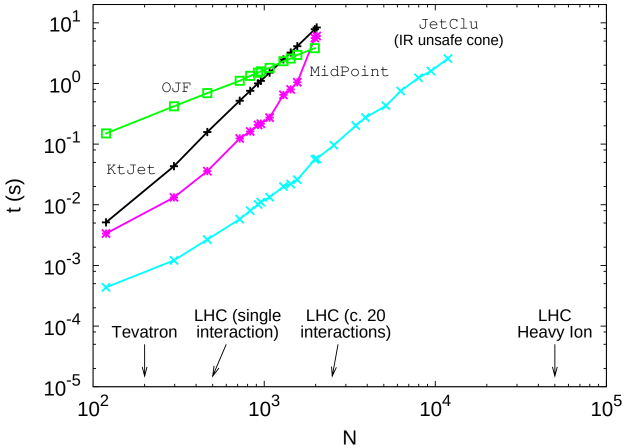
IR-safe cone (Mid-point) is as slow as k_t

Jet-clustering speed is an issue for high-luminosity *pp* ($\sim 10^8$ events) and Pb-Pb ($\sim 10^7$ events) collisions at LHC.

NB: want to rerun jet-alg. with a range of parameter choices

+ want to run on multiple MC samples of similar size

Time to cluster N particles



Standard C++ (and fortran) k_t-clustering takes time $\sim N^3$.

a Pb-Pb event takes 1 day!

JetClu (cone) *is fast*. But IR unsafe.

Discontinued at Tevatron

IR-safe cone (Mid-point) is as slow as k_t

Jet-clustering speed is an issue for high-luminosity *pp* ($\sim 10^8$ events) and Pb-Pb ($\sim 10^7$ events) collisions at LHC.

NB: want to rerun jet-alg. with a range of parameter choices

\+ want to run on multiple MC samples of similar size

How to make clustering faster?

Have $\mathcal{O}(N^2)$ distances d_{ij} to calculate. Is N^2 the best that can be done?

Problem of finding smallest d_{ij}

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2 \qquad \text{[kt]}$$

$$d_{ij} = \Delta R_{ij}^2 \qquad \text{[Cambridge/Aachen]}$$

can be **separated** into a momentum dependent part (k_t's) and geometrical part (ΔR_{ij}). Cacciari & GPS '05

- mom.-dependent part depends on just one particle, $\mathcal{O}(N)$ complexity
- geometrical parts ⇔ proximity problems widely studied by **computational geometers** → calculate only $\mathcal{O}(N \ln N)$ ΔR_{ij}'s
  Dynamic Voronoi diagrams: Devillers '99 (and many others) / CGAL
  Dynamic closest pair maintenance (quad-trees + shuffles): Chan '02

Put together in a C++ code — **FastJet**

Cacciari & GPS '05-'06

How to make clustering faster?

Have $\mathcal{O}\left(N^2\right)$ distances d_{ij} to calculate. Is N^2 the best that can be done?

Problem of finding smallest d_{ij}

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2 \qquad \text{[kt]}$$

$$d_{ij} = \Delta R_{ij}^2 \qquad \text{[Cambridge/Aachen]}$$

can be **separated** into a momentum dependent part (k_t's) and geometrical part (ΔR_{ij}). Cacciari & GPS '05

- mom.-dependent part depends on just one particle, $\mathcal{O}(N)$ complexity
- geometrical parts ⇔ proximity problems widely studied by **computational geometers** → calculate only $\mathcal{O}(N \ln N)$ ΔR_{ij}'s

Dynamic Voronoi diagrams: Devillers '99 (and many others) / CGAL

Dynamic closest pair maintenance (quad-trees + shuffles): Chan '02

Put together in a C++ code — **FastJet**

Cacciari & GPS '05-'06

How to make clustering faster?

Have $\mathcal{O}\left(N^2\right)$ distances d_{ij} to calculate. Is N^2 the best that can be done?

Problem of finding smallest d_{ij}

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2 \qquad \text{[kt]}$$

$$d_{ij} = \Delta R_{ij}^2 \qquad \text{[Cambridge/Aachen]}$$

can be **separated** into a momentum dependent part (k_t's) and geometrical part (ΔR_{ij}). Cacciari & GPS '05

- mom.-dependent part depends on just one particle, $\mathcal{O}(N)$ complexity
- geometrical parts ⇔ proximity problems widely studied by **computational geometers** → calculate only $\mathcal{O}(N \ln N)$ ΔR_{ij}'s
  Dynamic Voronoi diagrams: Devillers '99 (and many others) / CGAL
  Dynamic closest pair maintenance (quad-trees + shuffles): Chan '02

Put together in a C++ code — **FastJet**

Cacciari & GPS '05-'06

How to make clustering faster?

Have $\mathcal{O}\left(N^2\right)$ distances d_{ij} to calculate. Is N^2 the best that can be done?

Problem of finding smallest d_{ij}

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2)\Delta R_{ij}^2 \qquad \text{[kt]}$$

$$d_{ij} = \Delta R_{ij}^2 \qquad \text{[Cambridge/Aachen]}$$

can be **separated** into a momentum dependent part (k_t's) and geometrical part (ΔR_{ij}). Cacciari & GPS '05

- mom.-dependent part depends on just one particle, $\mathcal{O}(N)$ complexity
- geometrical parts ⇔ proximity problems widely studied by **computational geometers** → calculate only $\mathcal{O}(N \ln N)$ ΔR_{ij}'s

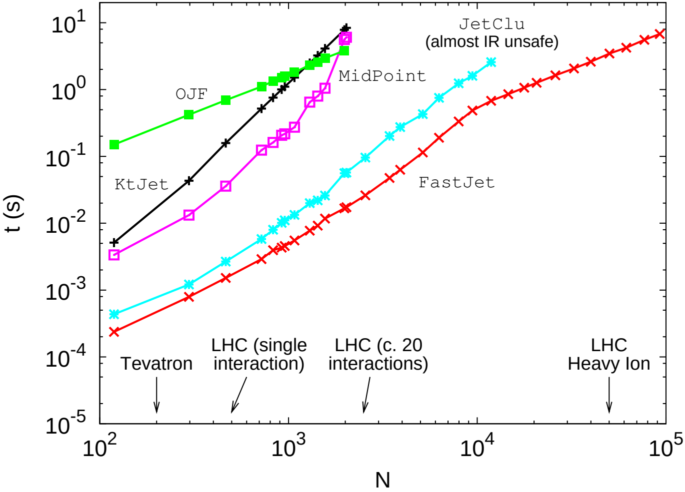
Dynamic Voronoi diagrams: Devillers '99 (and many others) / CGAL

Dynamic closest pair maintenance (quad-trees + shuffles): Chan '02

Put together in a C++ code — **FastJet**

Cacciari & GPS '05-'06

FastJet performance (kt)



For $N \gtrsim 10^4$, FastJet algorithm scales as $N \ln N$

For $N \lesssim 10^4$, FastJet switches to a related geometrical N^2 alg.

get code from http://www.lpthe.jussieu.fr/~salam/fastjet

Heavy ion background subtraction

Compared to low-lumi pp, crucial difference in Pb Pb is *huge* background.

Estimate of P_t density (ρ) at LHC:

$$\rho_{\text{background}} \equiv \frac{dP_t}{dy d\phi} \sim 250 \text{ GeV}$$

Hydjet 1.1 default, $dN_{\text{ch}}/dy = 1600 \quad y = 0$ (optimistic?)

Jet contamination:

$$\Delta P_{t,\text{jet}} \simeq \rho \times \text{Area}_{\text{jet}}$$

[Area $\sim \pi R^2$]

Correct before clustering

- by removing particles with $p_t < 1 - 2$ GeV
  Collinear unsafe; who knows how it's affected by quenching…
- by subtracting energy from calorimeter cells
  What to do with negative-energy cells? Experiment-dependent?

Correct after clustering

- Measure ρ and subtract $\rho \times \text{Area}_{\text{jet}}$.
  But what is jet area?

Heavy ion background subtraction

Compared to low-lumi pp, crucial difference in Pb Pb is *huge* background.

Estimate of P_t density (ρ) at LHC:

$$\rho_{\text{background}} \equiv \frac{dP_t}{dy d\phi} \sim 250 \text{ GeV}$$

Hydjet 1.1 default, $dN_{\text{ch}}/dy = 1600 \quad y = 0$ (optimistic?)

Jet contamination:

$$\Delta P_{t,\text{jet}} \simeq \rho \times \text{Area}_{\text{jet}}$$

[Area $\sim \pi R^2$]

Correct before clustering

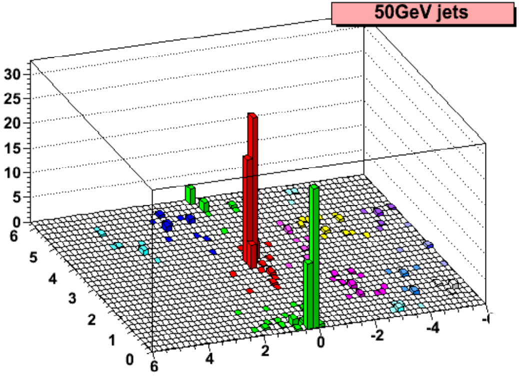
- by removing particles with $p_t < 1 - 2$ GeV
  Collinear unsafe; who knows how it's affected by quenching...
- by subtracting energy from calorimeter cells
  What to do with negative-energy cells? Experiment-dependent?

Correct after clustering

- Measure ρ and subtract $\rho \times \text{Area}_{\text{jet}}$.

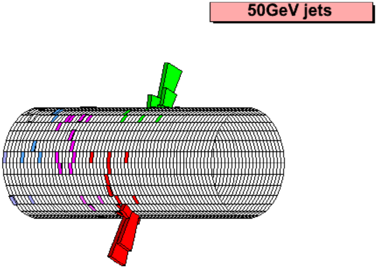
But what is jet area?

Jet areas (e.g. in pp with pileup)



'Standard hard' event
Two well isolated jets

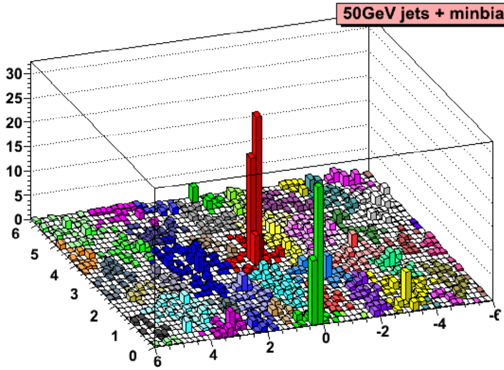
Jet boundaries completely unclear



$\sim$ 200 particles

Easy even with old methods

Jet areas (e.g. in pp with pileup)



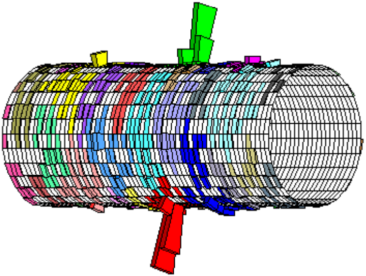
Add 10 min-bias events (moderately high lumi LHC pp)

Jet boundaries still ill defined — jets clearly irregular

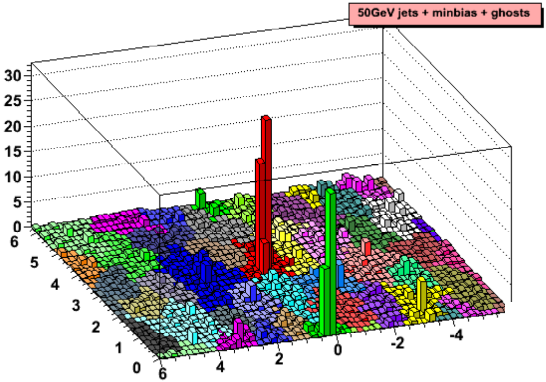
$\sim$ 2000 particles

Clustering takes $\mathcal{O}(10s)$ with old methods.

20ms with `FastJet`.



Jet areas (e.g. in pp with pileup)



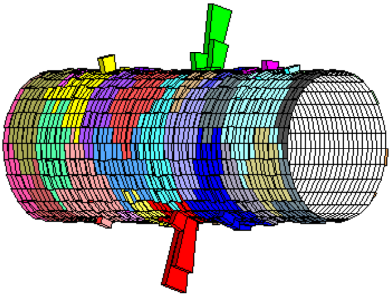
Add dense coverage of infinitely soft *"ghosts"*

See how many end up in jet to measure jet area

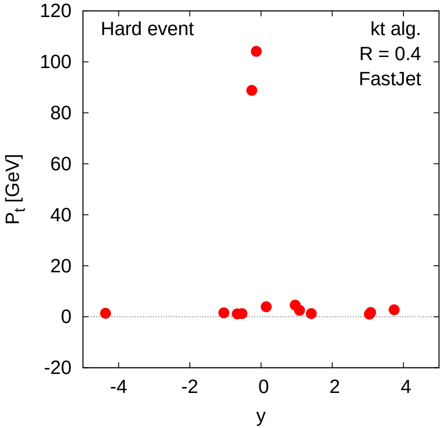
~ 10000 particles

Clustering takes ~ 20 minutes with old methods.

0.6s with `FastJet`.

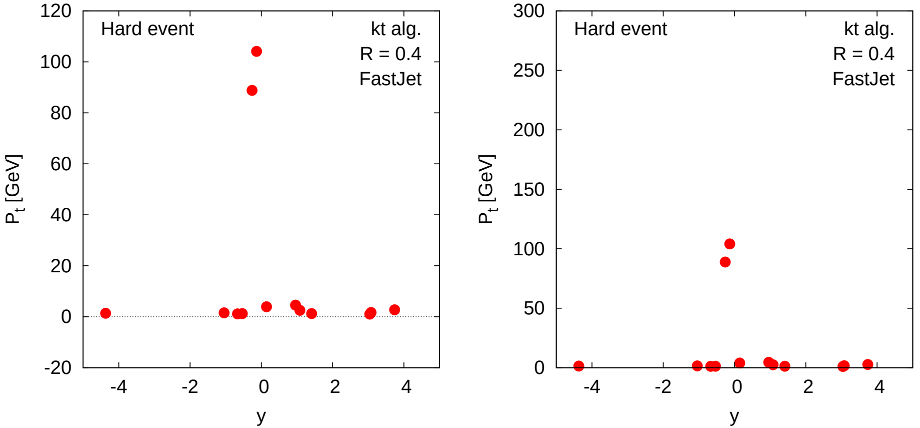


Background subtraction in HI event



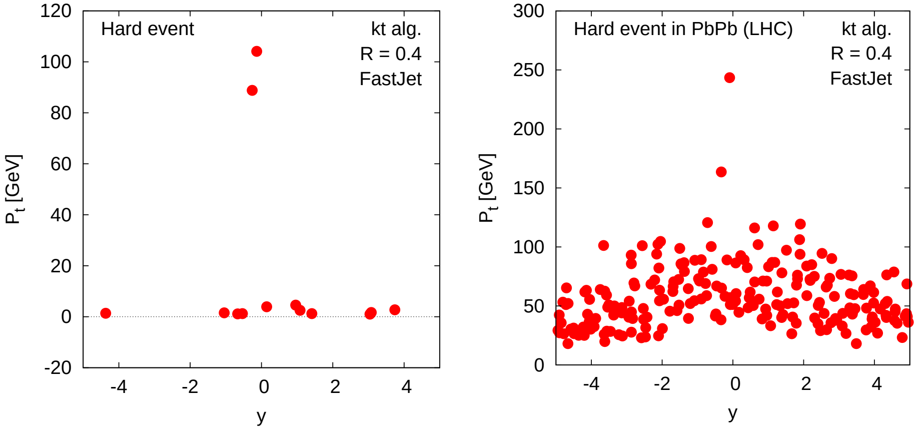
Start with a hard dijet event

Background subtraction in HI event



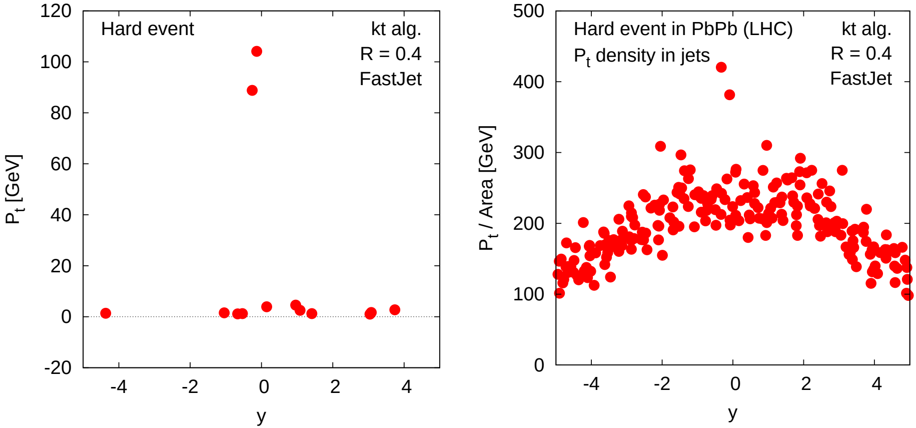
Same event on a different scale

Background subtraction in HI event



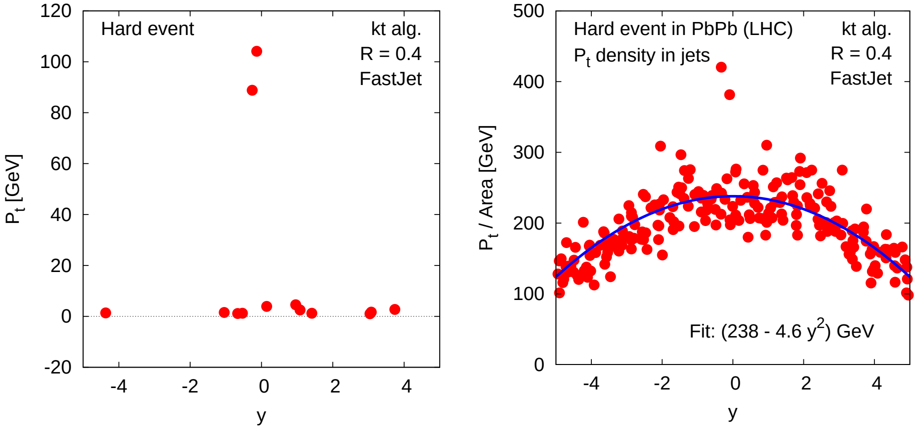
Embed it into a central Hydjet Pb Pb event

Background subtraction in HI event



Look at P_t/Area for each jet

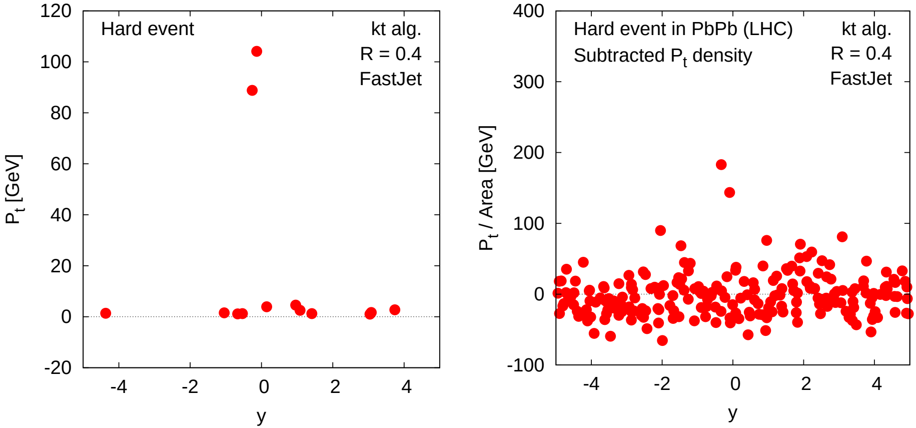
Background subtraction in HI event



Fit the background $\rho(y)$

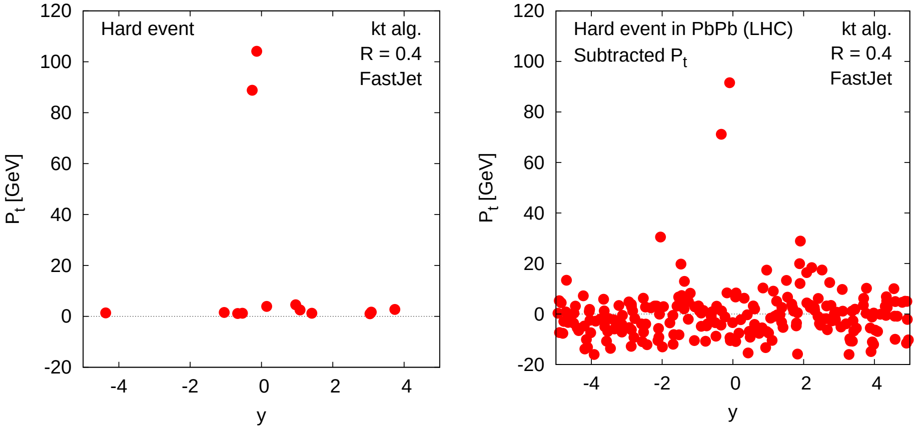
[NB: more general functional form needs investigating]

Background subtraction in HI event



Subtract $\rho(y)$ from P_t/Area for each jet

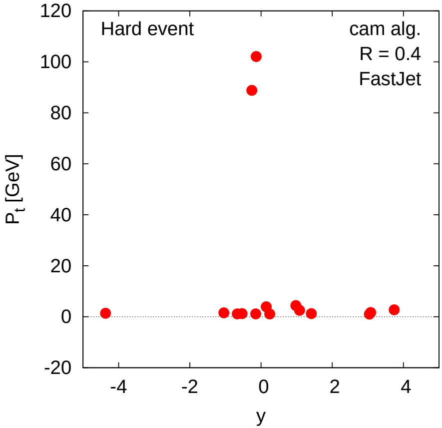
Background subtraction in HI event

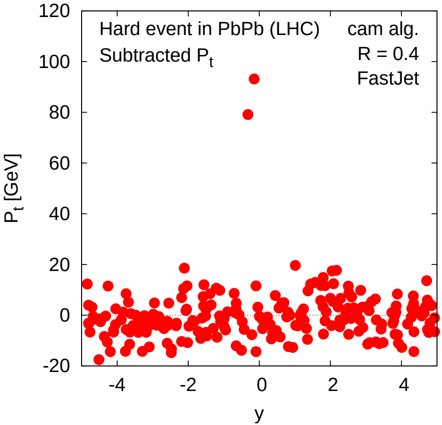


Look at resulting corrected $P_t = P_{t,orig} - \rho(y) \times \mathrm{Area}$

Hard jets with roughly correct P_t and y emerge clearly!

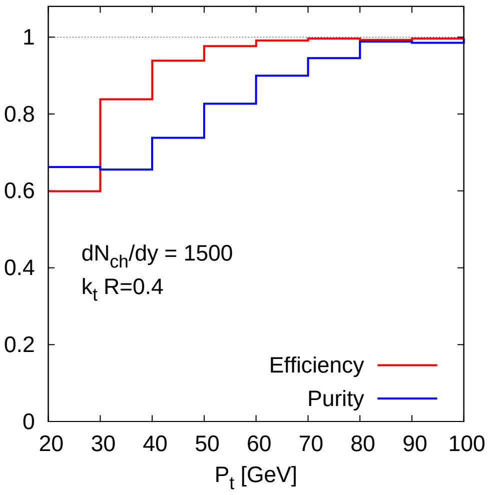
Background subtraction in HI event





Try with Cambridge/Aachen instead of k_t to check robustness!

Reconstruction efficiency & purity

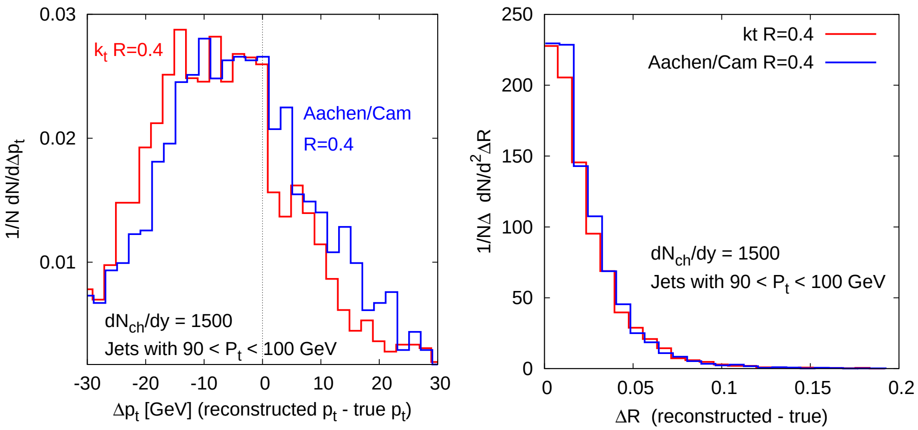


Procedure:

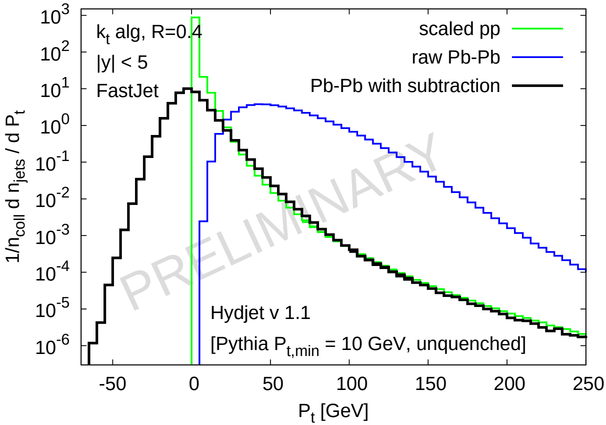
A reconstructed jet within $\Delta R < 0.2$ of the original jet is considered to correspond the original one.

NB: detector effects are likely to adversely affect these figures.

P_t & angular resolution



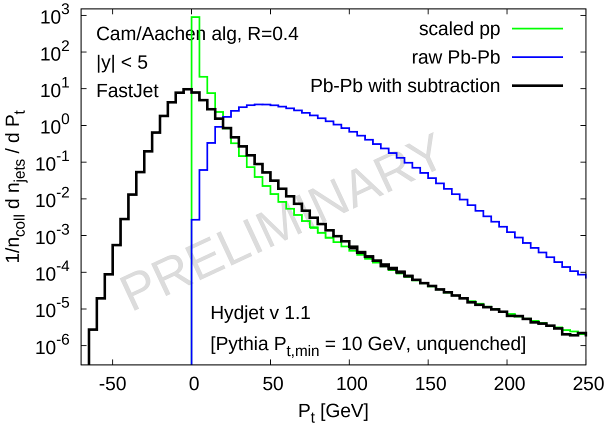
Inclusive jets in Pb-Pb @ LHC



Inclusive jet spectrum is most basic measurement in *pp*.

Interesting to check also in PbPb?

Inclusive jets in Pb-Pb @ LHC



Inclusive jet spectrum is most basic measurement in *pp*.

Interesting to check also in PbPb?

Conclusions

Clustering jet finders have attraction of **simplicity** when doing jet finding.

Important in complex environment like HI

Issues that "used to be", are no longer

- Speed: was N^3, now $N \ln N$; these are the fastest particle-level jet finders on the market!
- Ill-defined jet boundaries & area: add soft "ghosts" to track jet layout.
- **FastJet** code provides access to these tools.

Question of background subtraction is still open

- There are methods for doing it before clustering [ALICE, CMS, ATLAS], and after clustering [this talk].
- Preliminary studies show both to be effective — maybe interesting to combine strong points of each?

  In a standardized, collinear-safe, detector-independent formulation?
- Is there any chance of doing subtraction well enough for RHIC energies?

Conclusions

Clustering jet finders have attraction of **simplicity** when doing jet finding.

Important in complex environment like HI

Issues that "used to be", are no longer

- Speed: was N^3, now $N \ln N$; these are the fastest particle-level jet finders on the market!
- Ill-defined jet boundaries & area: add soft "ghosts" to track jet layout.
- **FastJet** code provides access to these tools.

Question of background subtraction is still open

- There are methods for doing it before clustering [ALICE, CMS, ATLAS], and after clustering [this talk].
- Preliminary studies show both to be effective — maybe interesting to combine strong points of each?

  In a standardized, collinear-safe, detector-independent formulation?
- Is there any chance of doing subtraction well enough for RHIC energies?

Conclusions

Clustering jet finders have attraction of **simplicity** when doing jet finding.

Important in complex environment like HI

Issues that "used to be", are no longer

- Speed: was N^3, now $N \ln N$; these are the fastest particle-level jet finders on the market!
- Ill-defined jet boundaries & area: add soft "ghosts" to track jet layout.
- **FastJet** code provides access to these tools.

Question of background subtraction is still open

- There are methods for doing it before clustering [ALICE, CMS, ATLAS], and after clustering [this talk].
- Preliminary studies show both to be effective — maybe interesting to combine strong points of each?

  In a standardized, collinear-safe, detector-independent formulation?
- Is there any chance of doing subtraction well enough for RHIC energies?

Conclusions

Clustering jet finders have attraction of **simplicity** when doing jet finding.

Important in complex environment like HI

Issues that "used to be", are no longer

- Speed: was N^3, now $N \ln N$; these are the fastest particle-level jet finders on the market!
- Ill-defined jet boundaries & area: add soft "ghosts" to track jet layout.
- **FastJet** code provides access to these tools.

Question of background subtraction is still open

- There are methods for doing it before clustering [ALICE, CMS, ATLAS], and after clustering [this talk].
- Preliminary studies show both to be effective — maybe interesting to combine strong points of each?

  In a standardized, collinear-safe, detector-independent formulation?
- Is there any chance of doing subtraction well enough for RHIC energies?

EXTRA MATERIAL

Why was k_t an N^3 algorithm?

1. Given the initial set of particles, construct a table of all the d_{ij}, d_{iB}.
   [$\mathcal{O}(N^2)$ *operations, done once*]
2. Scan the table to find the minimal value $d_{\min}$ of the d_{ij}, d_{iB}.
   [$\mathcal{O}(\mathbf{N^2})$ operations, done N times]
3. Merge or remove the particles corresponding to $d_{\min}$ as appropriate.
   [$\mathcal{O}(1)$ operations, done N times]
4. Update the table of d_{ij}, d_{iB} to take into account the merging or removal, and if any particles are left go to step 2.
   [$\mathcal{O}(N)$ operations, done N times]

This is the "brute-force" or "naive" method

Can we do better than N^2?

There are $N(N-1)/2$ distances d_{ij} — surely we have to calculate them all in order to find smallest?

k_t distance measure is partly *geometrical:*

- Consider smallest $d_{ij} = \min(k_{ti}^2, k_{tj}^2) R_{ij}^2$
- Suppose $k_{ti} < k_{tj}$
- Then: $R_{ij} <= R_{i\ell}$ for any $\ell \neq j$. [If $\exists\ \ell$ s.t. $R_{i\ell} < R_{ij}$ then $d_{i\ell} < d_{ij}$]

In words: if i, j form smallest d_{ij} then j is geometrical nearest neighbour (GNN) of i.

k_t distance need only be calculated between GNNs

Each point has 1 GNN → need only calculate N d_{ij}'s

Can we do better than N^2?

There are $N(N-1)/2$ distances d_{ij} — surely we have to calculate them all in order to find smallest?

k_t distance measure is partly *geometrical:*

- Consider smallest $d_{ij} = \min(k_{ti}^2, k_{tj}^2) R_{ij}^2$
- Suppose $k_{ti} < k_{tj}$
- Then: $R_{ij} <= R_{i\ell}$ for any $\ell \neq j$. [If $\exists\ \ell$ s.t. $R_{i\ell} < R_{ij}$ then $d_{i\ell} < d_{ij}$]

In words: if i, j form smallest d_{ij} then j is geometrical nearest neighbour (GNN) of i.

k_t distance need only be calculated between GNNs

Each point has 1 GNN $\rightarrow$ need only calculate N d_{ij}'s

Can we do better than N^2?

There are $N(N-1)/2$ distances d_{ij} — surely we have to calculate them all in order to find smallest?

k_t distance measure is partly *geometrical:*

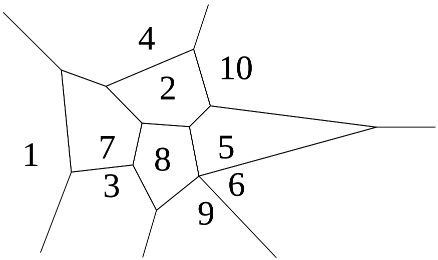
- Consider smallest $d_{ij} = \min(k_{ti}^2, k_{tj}^2) R_{ij}^2$
- Suppose $k_{ti} < k_{tj}$
- Then: $R_{ij} <= R_{i\ell}$ for any $\ell \neq j$. [If $\exists\ \ell$ s.t. $R_{i\ell} < R_{ij}$ then $d_{i\ell} < d_{ij}$]

In words: if i, j form smallest d_{ij} then j is geometrical nearest neighbour (GNN) of i.

k_t distance need only be calculated between GNNs

Each point has 1 GNN $\rightarrow$ need only calculate N d_{ij}'s

Finding Geom Nearest Neighbours



Given a set of vertices on plane (1...10) a *Voronoi diagram* partitions plane into cells containing all points closest to each vertex

Dirichlet '1850, Voronoi '1908

A vertex's nearest other vertex is always in an adjacent cell.

E.g. GNN of point 7 will be found among 1,4,2,8,3 (it turns out to be 3)

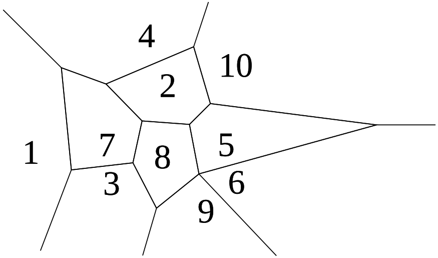
Construction of Voronoi diagram for N points: $N \ln N$ time   Fortune '88

Update of 1 point in Voronoi diagram: $\ln N$ time

Devillers '99 [+ related work by other authors]

Convenient C++ package available: **CGAL**   http://www.cgal.org

Finding Geom Nearest Neighbours



Given a set of vertices on plane (1...10) a *Voronoi diagram* partitions plane into cells containing all points closest to each vertex

Dirichlet '1850, Voronoi '1908

A vertex's nearest other vertex is always in an adjacent cell.

E.g. GNN of point 7 will be found among 1,4,2,8,3 (it turns out to be 3)

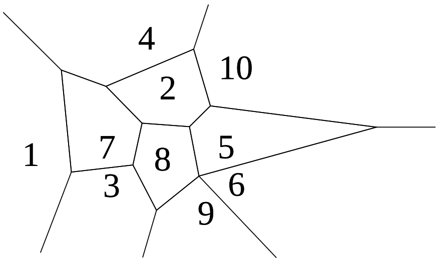
Construction of Voronoi diagram for N points: $N \ln N$ time Fortune '88

Update of 1 point in Voronoi diagram: $\ln N$ time

Devillers '99 [+ related work by other authors]

Convenient C++ package available: **CGAL** http://www.cgal.org

Finding Geom Nearest Neighbours



Given a set of vertices on plane (1...10) a *Voronoi diagram* partitions plane into cells containing all points closest to each vertex

Dirichlet '1850, Voronoi '1908

A vertex's nearest other vertex is always in an adjacent cell.

E.g. GNN of point 7 will be found among 1,4,2,8,3 (it turns out to be 3)

Construction of Voronoi diagram for N points: $N \ln N$ time Fortune '88

Update of 1 point in Voronoi diagram: $\ln N$ time

Devillers '99 [+ related work by other authors]

Convenient C++ package available: **CGAL** http://www.cgal.org

Assembling fast k_t clustering

The FastJet algorithm:

Construct the Voronoi diagram of the N particles with CGAL **$\mathcal{O}(N \ln N)$**

Find the GNN of each of the N particles, calculate d_{ij} store result in a *priority queue* (`C++ map`) **$\mathcal{O}(N \ln N)$**

Repeat following steps **N** times:

- Find smallest d_{ij}, merge/eliminate i, j **$N \times \mathcal{O}(1)$**
- Update Voronoi diagram and distance map **$N \times \mathcal{O}(\ln N)$**

Overall an $\mathcal{O}(N \ln N)$ algorithm

Cacciari & GPS, hep-ph/0512210

http://www.lpthe.jussieu.fr/~salam/fastjet/

Results identical to standard N^3 implementations

Assembling fast k_t clustering

The `FastJet` algorithm:

Construct the Voronoi diagram of the N particles with CGAL **$\mathcal{O}(N \ln N)$**

Find the GNN of each of the N particles, calculate d_{ij} store result in a *priority queue* (`C++ map`) **$\mathcal{O}(N \ln N)$**

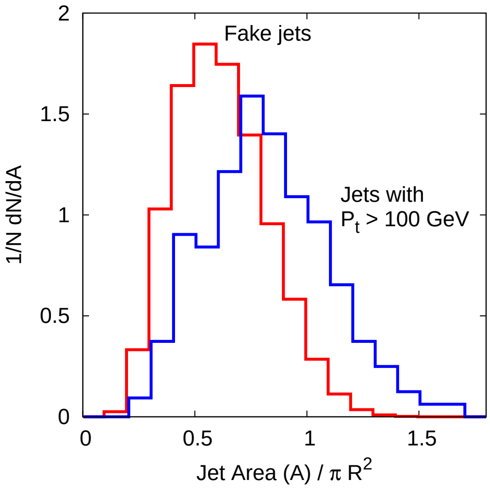
Repeat following steps **N** times:

- Find smallest d_{ij}, merge/eliminate i, j **$N \times \mathcal{O}(1)$**
- Update Voronoi diagram and distance map **$N \times \mathcal{O}(\ln N)$**

Overall an $\mathcal{O}(N \ln N)$ algorithm

Cacciari & GPS, hep-ph/0512210
http://www.lpthe.jussieu.fr/~salam/fastjet/
Results identical to standard N^3 implementations

Jet areas vary



Each jet has a different area

True jets can have internal structure (parton branching) — jet area expands to accomodate this.

Fake jets little internal structure $\rightarrow$ jet areas smaller.

NB: jet areas often $< \pi R^2$