FastJet & FJContrib

Matteo Cacciari (Paris)
Gavin Salam (CERN)
Gregory Soyez (Saclay)
22 additional FJContrib authors
Lines of code - 1 year ago

FastJet
17k lines of code
16k lines of comments

fjcore
6k lines

fjcontrib
6 contribs
3k lines
2k comments

FastJet 3.0.4
fjcontrib 1.005
Lines of code - today

**FastJet**
- 20k lines of code
- 19k lines of comments

**fjcore**
- 8k lines of code

**fjcontrib**
- 12 contribs
- 9k lines of code
- 5k comments

FastJet 3.1.0-beta.1

fjcontrib 1.014
FastJet 3.1

β.1 released last Friday
New things include
❖ speed gains
❖ hadron masses in area subtraction
❖ facilities of help in FJContrib
FastJet chooses different code variants, 
“strategies” 
for clustering based on event multiplicity & R

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2Plain</td>
<td>a plain $N^2$ algorithm (fastest for $N \lesssim 30$)</td>
</tr>
<tr>
<td>N2Tiled</td>
<td>a tiled $N^2$ algorithm (fastest for $30 \lesssim N \lesssim 400$)</td>
</tr>
<tr>
<td>N2MinHeapTiled</td>
<td>a tiled $N^2$ algorithm with a heap for tracking the minimum of $d_{ij}$ (fastest for $400 \lesssim N \lesssim 15000$)</td>
</tr>
<tr>
<td>NlnN</td>
<td>the Voronoi-based $N \ln N$ algorithm (fastest for $N \gtrsim 15000$)</td>
</tr>
<tr>
<td>NlnNCam</td>
<td>based on Chan’s $N \ln N$ closest pairs algorithm (fastest for $N \gtrsim 6000$), suitable only for the Cambridge jet algorithm</td>
</tr>
<tr>
<td>Best</td>
<td>automatic selection of the best of these based on $N$ and $R$</td>
</tr>
</tbody>
</table>

From FastJet 3.0 manual
FastJet lemma (recall)

You need to find smallest of the $N^2 d_{ij}$

$$d_{ij} = \max(p_{t_i}^{-2}, p_{t_j}^{-2}) \Delta R_{ij}^2$$

FJ never looks through all $N^2 d_{ij}$, but instead exploits lemma that for given particle $i$, smallest $d_{ij}$ must come from i’s geometrical nearest neighbour
N2Plain strategy

Look for geometrical nearest neighbour (GNN) among all $N$ particles.

*Why is this fast?*
Because particle $i$ is GNN of only $O(1)$ other particles; so when you remove it, updates of other particles’ GNNs costs $O(N) \rightarrow O(N^2)$ total time.
Particles only cluster with others within $\Delta R < R$

Arrange particles on grid of spacing $\sim R$. Look for geometrical nearest neighbour (GNN) within 3x3 group of tiles.

Gives alg that’s $O(Nn)$

$n$ is # of particles in a tile + grid setup overhead.
New: N2MHT Lazy9

1. Look in 1 tile for GNN

2. Consider only surrounding tiles whose edge is closer than in-tile GNN

Still $O(Nn)$, but with a smaller coefficient at high densities.

[Price of extra bookkeeping compensated by smaller # of tiles to search through]
New: N2MHT Lazy25

Like Lazy9 but instead of 3x3 neighbourhood of tiles of size R, use 5x5 neighbourhood of tiles of size $\frac{1}{2}R$.

Some overheads grow, but at high densities, net gain from reduced area over which to search for GNN.
New: N2MHTLazy25

Like Lazy9 but instead of 3x3 neighbourhood of tiles of size $R$, use 5x5 neighbourhood of tiles of size $\frac{1}{2}R$.

Some overheads grow, but at high densities, net gain from reduced area over which to search for GNN.
“N \ln N” \quad \text{(CGAL Voronoi)}

\textbf{Original FastJet idea}

Restrict GNN search to nearby Voronoi cells:

- typically only $O(10)$ to search through \textit{except for anti-k_t}
- but high coefficient of $\ln N$ in order to maintain Voronoi diagram

NB: for anti-k_t, alg is $N^{3/2}$

FJ 3.1 fixes a coincident point issue
How do they compare?

Time to cluster $N$ particles (per thousand particles)

Shown here for $R=1.2$
How do they compare?

Time to cluster $N$ particles (per thousand particles)

Shown here for $R=1.2$

Gain starts for $N=500$, largest around 10k
Automated strategy choice

Based on events with particles up to |y| = 5

region where new strategies are optimal
3.0 → 3.1 speed gains (R=0.4)

Time to cluster N particles

Improvement wrt FJ 3.0.x, factor of 2 for 10k
3.0 → 3.1 speed gains (R=1.2)

Time to cluster N particles

Improvement wrt FJ 3.0.x, factor of 6 for 10k
Other speed “things”

- **N2MHTLazy9AntiKtSeparateGhosts**
  Anti-$k_t$ only, clusters ghosts separately (but no ghost jets)
  Still preliminary, but worth looking at if speed matters

- Automated strategy choice not optimal for jet reclustering

- There may still be room for improvement for large $R$, large $N$
PU subtraction & jet masses

New facilities in FJ3.1
The wisdom of including hadron masses
PU subtraction & hadron masses

- FastJet 3.0 provides you with $\rho = p_t$ per unit area.

- If your “hadrons” have masses, you also need $\rho_m, m_\delta$ per unit area: [Soyez et al, 1211.2811]

$$m_\delta = \sum_{i \in \text{area}} \left( \sqrt{m_i^2 + p_{t,i}^2} - p_{ti} \right)$$

- Subtraction then has extra longitudinal terms

$$p_{\text{jet,sub}}^\mu = p_{\text{jet}}^\mu - \left[ \rho A^x_{\text{jet}}, \rho A^y_{\text{jet}}, (\rho + \rho_m) A^z_{\text{jet}}, (\rho + \rho_m) A^E_{\text{jet}} \right]$$

In FJ 3.1

- BackgroundEstimators have new `bge.rho_m()` method

- **Enable** its use in Subtractors with `subtractor.set_use_rho_m()`
Illustration

Start with W peak & QCD continuum
Start with W peak & QCD continuum
❖ Add pileup
Illustration

Start with W peak & QCD continuum

❖ Add pileup
❖ Subtract without \( \rho_m \)
Illustration

Start with W peak & QCD continuum

❖ Add pileup
❖ Subtract without $\rho_m$
❖ Instead subtract with $\rho_m$

W peak is back where it should, though very smeared out
Massless v. massive hadrons

There is perhaps no good reason for including hadron masses
negative jet masses

Unphysical negative $m^2$
Give negative mass in FJ
positive jet masses

This can have a noticeable effect on performance plots

```
subtractor.set_safe_mass()  # forces m ≥ 0
```
FJContrib

9 releases since last Boost
6 new contribs
25 contributors

3rd party extensions to FastJet
Version 1.005 of FastJet Contrib is distributed.

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenericSubtractor</td>
<td>1.2.0</td>
</tr>
<tr>
<td>JetFFMoments</td>
<td>1.0.0</td>
</tr>
<tr>
<td>VariableR</td>
<td>1.0.1</td>
</tr>
<tr>
<td>Nsubjettiness</td>
<td>1.0.2</td>
</tr>
<tr>
<td>EnergyCorrelator</td>
<td>1.0.1</td>
</tr>
<tr>
<td>ScJet</td>
<td>1.1.0</td>
</tr>
</tbody>
</table>

Version 1.014 of FastJet Contrib is distributed.

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConstituentSubtractor</td>
<td>1.0.0</td>
</tr>
<tr>
<td>EnergyCorrelator</td>
<td>1.0.1</td>
</tr>
<tr>
<td>GenericSubtractor</td>
<td>1.2.0</td>
</tr>
<tr>
<td>JetCleanser</td>
<td>1.0.1</td>
</tr>
<tr>
<td>JetFFMoments</td>
<td>1.0.0</td>
</tr>
<tr>
<td>JetsWithoutJets</td>
<td>1.0.0</td>
</tr>
<tr>
<td>Nsubjettiness</td>
<td>2.1.0</td>
</tr>
<tr>
<td>RecursiveTools</td>
<td>1.0.0</td>
</tr>
<tr>
<td>ScJet</td>
<td>1.1.0</td>
</tr>
<tr>
<td>SoftKiller</td>
<td>1.0.0</td>
</tr>
<tr>
<td>SubjetCounting</td>
<td>1.0.1</td>
</tr>
<tr>
<td>VariableR</td>
<td>1.1.1</td>
</tr>
</tbody>
</table>
FJ 3.1 facilities for fjcontrib etc.

Recluster
RectangularGrid
FASTJET_VERSION_NUMBER
It’s easy enough to recluster the particles in a jet.

```cpp
JetDefinition jet_def(cambridge_algorithm, 1000.0);
PseudoJet reclustered_jet = jet_def(jet.constituents())[0];
```

But what if the jet had areas? A non-standard recombiner?

```cpp
// simplified illustration of handling of areas
const ClusterSequence * cs = jet.associated_cluster_sequence();
if (cs) {
  if (cs->has_explicit_ghosts()) {
    vector<PseudoJet> particles, ghosts;
    SelectorIsPureGhost.sift(jet.constituents(), ghosts, particles);
    double ghost_area = ghosts[0].area(); //+ case without ghosts
    auto csa = new ClusterSequenceActiveAreaExplicitGhosts(
      particles, jet_def, ghosts, ghost_area);
    reclustered_jet = SelectorNHardest(1)(csa->inclusive_jets());
    csa->delete_self_when_unused();
  }
}
```
Recluster

It’s easy enough to recluster the particles in a jet

```cpp
JetDefinition jet_def(cambridge_algorithm, 1000.0);
PseudoJet reclustered_jet = jet_def(jet.constituents())[0];
```

But what if the jet had areas? A non-standard recombiner? Quickly becomes painful. Instead use Recluster:

```cpp
// Recluster looks at the input jet and automatically
// - reclusters with areas if it detects (explicit) ghosts
// - reclusters with the original jet’s recombiner
// - looks into pieces to see if they share a CS
#include “fastjet/tools/Recluster.hh”

Recluster recluster_CA(cambridge_algorithm);
PseudoJet reclustered_jet = recluster_CA(jet);
```

Also exploits new FJ31 facility: `jet_def_B.set_recombiner(jet_def_A)`
RectangularGrid

e.g. GridMedianBackgroundEstimator
GridJet, SoftKiller

New class gives common interface, more flexible grid layout

```
#include “fastjet/RectangularGrid.hh”

// can specify asymmetric rap lims, separate y & phi spacings
double ymin = -5.0, ymax=-2.0, dy=0.5, dphi = twopi/12;
RectangularGrid lhcb_grid(ymin, ymax, dy, dphi);
SoftKiller soft_killer(lhcb_grid);

// facility to remove subset of tiles from grid
Selector not_central = !SelectorRapRange(-2.5,2.5);
RectangularGrid forward_grid(-5.0, 5.0, dy, dphi, not_central);
GridMedianBackgroundEstimator forward_bge(forward_grid);
```
FJ version detection

E.g. you want your new contrib to exploit FJ3.1 facilities if available, but also stay compatible with FJ3.0.

```cpp
#include "fastjet/config.h"

// version xx.yy.zz has FASTJET_VERSION_NUMBER = XXYYZZ
// e.g. test for version >= 3.1.0
#if FASTJET_VERSION_NUMBER >= 30100
#include "fastjet/RectangularGrid.hh"
#endif

class MyNewContrib {
    // provide constructor only when used with FJ3.1 and higher
#if FASTJET_VERSION_NUMBER >= 30100
    MyNewContrib(const RectangularGrid & grid);
#endif
```
Outlook
FastJet

Our aim is to concentrate FJ development on core features.

- Next major milestone is **thread safety**
- Is there scope for further speed improvement?
  (At least in terms of strategy selection)
Our experience so far is positive. What is yours?

Issues that we see include:

❖ Dependencies (between contribs, on external libs)
❖ Shared library support
❖ Review of new contribs & updates is getting slow (we are short of time; insightful feedback takes time)
❖ Long-term maintenance for a “distributed” project. If a tool is useful it may stay in use for 10–20 years.
Backup slides
How do they compare?

Time to cluster $N$ particles (per thousand particles)

Shown here for $R=0.4$

Gain starts for $N=2k$, largest around 100k