LPTHE-06-02
March 2006

Jet clustering in particle physics, via a dynamic nearest neighbour graph implemented with CGAL

Gavin P. Salam and Matteo Cacciari
LPTHE, Université Pierre et Marie Curie – Paris 6,
Université Denis Diderot – Paris 7, CNRS UMR 7589,
75252 Paris 75005, France
e-mail: {salam,cacciari}@lpthe.jussieu.fr

Abstract

High-energy collisions in particle physics experiments produce complex events with large numbers, N, of particles. A widely advocated approach to the analysis of these events involves an agglomerative clustering procedure (known as the k_t jet finder) to identify clusters of particles, jets, which have a direct connection with the underlying particle physics reaction. The clustering uses a distance measure that can be factorised into geometrical and non-geometrical components, allowing the problem to be reduced to the dynamic maintenance, across $\mathcal{O}(N)$ updates, of a planar nearest-neighbour graph and a priority queue. The former can be implemented with CGAL's hierarchical Delaunay triangulation module, allowing the clustering to be carried out in expected (and measured) $\mathcal{O}(N \log N)$ time, a practically important improvement relative to the $\mathcal{O}\left(N^3\right)$ of the previously standard brute-force approach.

1 Introduction

Particle physics [4, 47] seeks to identify the fundamental constituents of matter and understand the interactions between them. Currently several classes of elementary particles are known: leptons, such as electrons and neutrinos; quarks, which are the constituents of protons and neutrons, and vector bosons, which mediate forces between these particles, such as photons (responsible for the electromagnetic force), W and Z bosons (the weak force) and gluons (the strong force).

In order to discover new very massive particles or establish whether the above particles might be composite, i.e. made of some other yet smaller, more fundamental particles, particle physicists carry out experiments in which they collide beams of high energy electrons or protons. By virtue of Einstein's famous $E = mc^2$ relation between energy E, and mass m (with c the speed of light), a high collision energy makes it possible to probe high mass scales. At the same time, because of the Heisenberg uncertainty principle ($\Delta x \Delta p \simeq \hbar/2$,

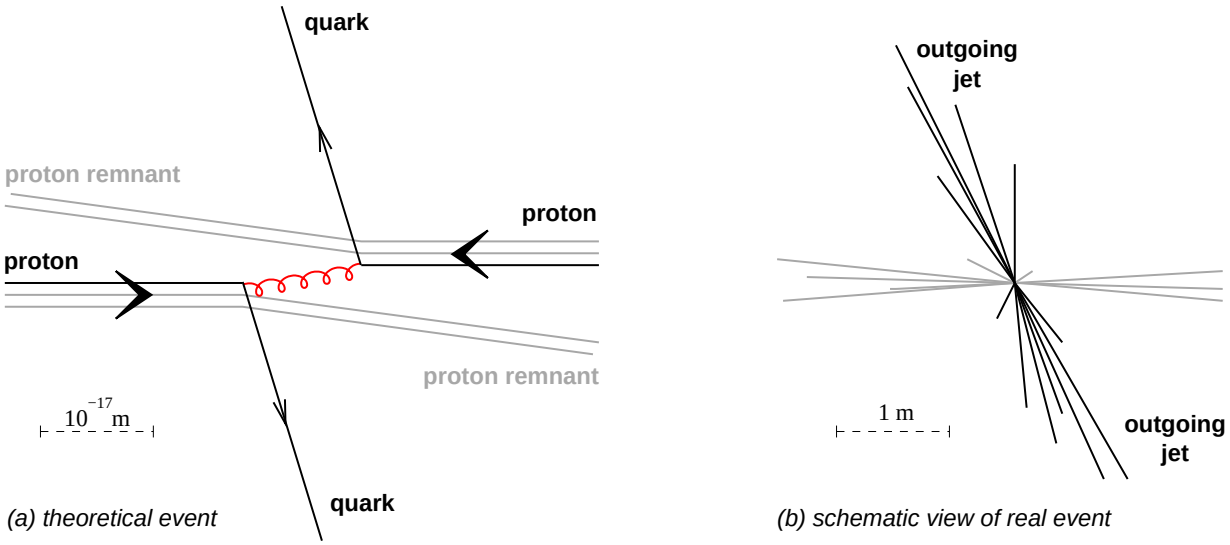

Figure 1: (a) Theoretical interpretation of a proton-proton collision: each proton consists of three quarks and in the collision there is a large exchange of momentum (mediated by the gluon, the curly line) between a quark in each proton, causing them to be strongly deviated; (b) a schematic representation of a real event as seen in a particle physics detector.

where Δx represents a length, Δp a transferred momentum and $\hbar$ is Planck's constant divided by 2π), a large energy (i.e. possibly large momentum transfer) means that extremely small length scales can be probed.

A typical high-energy physics collision is represented in fig. 1a, in which two protons (each made of three quarks) collide and a gluon transfers a large amount of momentum from a quark in one proton to a quark in the other proton. These two quarks are strongly deviated, exiting the collision region at large angle, while the other remnants of the two protons carry on essentially in their original direction. The distributions for the energy and angle of the scattered quarks can be predicted theoretically. To test whether the theory is complete, or whether instead some other as yet unknown particle can be exchanged in addition to the gluon, one needs to compare the theoretically predicted distribution to experimental measurements.

Though reactions such as those in fig. 1a take place on extremely small length scales ($\ll 10^{-15}$ m), the actual measurements of the results of collision are carried with sophisticated multi-layered detectors built on human scales, roughly $1\,\text{cm} - 10\,\text{m}$. On these scales the collision looks like fig. 1b, with each quark having been replaced by a multitude $(10 - 100)$ of 'hadrons' (objects, such as such as protons, neutrons and pions, that are composed of quarks, antiquarks and gluons). The reason for this is that quarks and gluons, as they leave the collision point, start to fragment: initially they emit gluons, which can split into quark-antiquark pairs, or themselves emit further gluons, and so forth — in other words there is a cascade of emissions, first at small distance scales, then at progressively larger distances, up to about 10^{-15} m. Beyond this scale a transition takes place in which hadrons

are formed from the quarks and gluons, and it is these hadrons that are observed in the detectors.

The theory that determines the pattern of quark-quark scattering and the subsequent cascading is known as Quantum Chromodynamics (QCD) [3]. Though it can be summarised in one main equation, finding solutions to that equation is often extremely complex. In particular, while the initial quark-quark scattering in fig. 1a can be well predicted, and there is some reasonable understanding of the cascading to produce multiple quarks and gluons, there is currently no first-principles understanding of how to calculate the transition from quarks and gluons to hadrons. However, in a seminal paper in 1977, Sterman and Weinberg were able to show [19] that one could ignore much of the cascading and the transition to hadrons, if instead of examining individual particles, one clusters bunches of particles together as 'jets' and examines the distributions in energy and angle of these jets. This idea, applied both to the theoretical predictions and to the experimental measurements, has been enormously successful [34].

The original proposal for finding jets was based on the identification of cones which contain most of the energy-momentum flow in the event. Since then approaches that relate to those in the general pattern classification literature [2] have been also investigated, notably K-means (the so called 'optimal' jet finder [20]) and hierarchical agglomerative clustering (the Jade [6], Durham/k_t [10], Cambridge/Aachen [13] and flavour [37] jet finders). In agglomerative clustering one introduces a distance measure between particles, searches for the closest pair of particles, merges them into a single particle, and then repeats the procedure until all pair distances are larger than some threshold. Because the cascade that produced the multitude of particles in the first place was a form of hierarchical branching, agglomerative clustering is particularly suited for trying to reconstruct the 'original' particle, since it works 'backwards' through the branching. For this reason it has been the preferred jet-finding procedure in the majority of recent experiments, in particular in electron-positron collisions at the Large Electron Positron collider (LEP, CERN, Geneva) and the Stanford Linear Collider (SLC, SLAC, Stanford), and in electron-proton collisions at HERA (DESY, Hamburg).

In some experiments, notably at the Fermilab Tevatron proton-antiproton collider (Batavia, Illinois) it is still a variant of the original 'cone-based' jet finding procedure that is more commonly used (except in [25, 43]). The reasons for this are complex, however one relevant issue is that collisions at the Tevatron tend to produce larger numbers (N) of particles than at other colliders[1] (a few hundred rather than about fifty, and the effective number can increase to several thousand depending on the manner in which the detector information is read out). This is a problem because the codes for agglomerative jet clustering [17] all use a brute force algorithm, which scales as N^3. It will become a much more severe issue at the upcoming Large Hadron Collider, a proton-proton collider due to start operating at CERN (Geneva) in 2007: N will be larger there both because of

[1]Since both colliding beams at the Tevatron are (anti)protons, these interact via the strong force (QCD) and produce many particles, whereas other recent experiments have electrons (or positrons) in at least one of the beams, and these do not interact via the strong force.

a sevenfold increase in energy relative to the Tevatron (giving a factor of two in N) and because there will be multiple simultaneous interactions (a factor of $10 - 20$ in N). The LHC will also collide lead ions, leading to $N \sim 50000$ (a similar experiment, RHIC, at lower energy, is currently running at Brookhaven National Laboratory, on Long Island). Overall one expects datasets of about $\sim 10^9$ events with a few thousand particles [36] and $\sim 10^7$ events with many tens of thousands of particles [44], which translates to $10^2 - 10^4$ CPU years with the brute force algorithms. Processing time is therefore a central issue.

According to a survey in an article by Eppstein [14], the use of the brute force algorithm for agglomerative clustering is not uncommon, especially in situations where, as for jet finding, the distance measure is non-geometrical. Better procedures are known in such cases [1, 14, 42] however at best they are $\mathcal{O}(N^2)$.

In the specific context of the jet finding we recently observed in a short letter [40][2] that for the most commonly used distance measure in agglomerative jet finding, the longitudinally invariant k_t distance [10], it is possible to separate the problem into a part involving a priority queue and another involving a nearest-neighbour graph on a cylinder (which in turn can be reduced to a plane). Both need to be kept up to date with respect to deletions and insertions as the clustering proceeds through its $\mathcal{O}(N)$ steps.

Priority queues are sufficiently widely-used that they are provided in the C++ Standard Template Library; their construction and maintenance during the clustering will be associated with a cost of $\mathcal{O}(N \log N)$.

Dynamic planar nearest neighbour graphs in contrast are a subject of current research. It is still not known [46] if there exists an algorithm that allows for $\mathcal{O}(\log N)$ worst case time per update and per query for the simpler but related problem of nearest-neighbour finding in a dynamic set of points. Recalling that the nearest-neighbour graph is a subgraph of the Delaunay Triangulation (DT), here we shall investigate the use of the triangulations component [9] of CGAL [15], which provides a fully dynamic Delaunay Triangulation, based on the hierarchical structure of Devillers [23, 24]. Updates to the hierarchy take expected $\mathcal{O}(\log N + k \log k)$ time where k is the degree of the vertex being updated [23], which reduces to expected $\mathcal{O}(\log N)$ insofar as k is expected $\mathcal{O}(1)$ for our application. Thus, given that there are $\mathcal{O}(N)$ updates to the DT hierarchy during clustering, the clustering will run in expected $\mathcal{O}(N \log N)$ time.

The structure of this paper is as follows. In section 2 we shall present the longitudinally invariant k_t clustering jet finder, and discuss briefly the brute force algorithm that is currently in use, as well as more efficient algorithms [1, 14, 42] for agglomerative clustering with general distance measures. In section 3 we shall then show how the problem reduces to a priority queue and dynamic maintenance of a nearest-neighbour graph, and examine current approaches, in particular CGAL, for dealing with the second of these problems. In section 4 we shall consider how the CGAL based solution performs in practice, comparing it both to the original N^3 brute force method and to N^2 methods for dealing with the nearest-neighbour graph.

[2] Aimed at the particle physics community, and on which this article elaborates.

2 The longitudinally invariant k_t jet-finder

2.1 Definition

The most widely studied agglomerative clustering jet finder for proton-(anti)proton collisions is the so-called longitudinally invariant k_t jet finder. Its name stems from the set of variables it uses for representing particle momenta and distances between them. To understand how the geometrical problem will arise later, we need to examine these kinematic variables (for simplicity we restrict ourselves to the approximation that particles are massless). A particle momentum vector $\vec{k}$ can be represented in polar coordinates, $\vec{k} = (k_x, k_y, k_z) = E/c \times (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$, where E is the particle's energy. The incoming (anti)protons enter conventionally along the $\pm z$ direction.

Though a polar coordinate system has the advantage of being familiar, it turns out that it is not the most natural for describing proton-proton collisions. The reason is related to special relativity, which implies that many (average) aspects of the collision should look the same to a 'laboratory-frame' observer and to one who is travelling fast alongside one of the protons (i.e. longitudinally). Some kinematic quantities are invariant with respect to such longitudinal frame changes, in particular the transverse momentum $k_t = \sqrt{k_x^2 + k_y^2}$ and the azimuthal angle ϕ. The polar angle in contrast transforms in a more complicated manner, so instead one considers a quantity called (pseudo)rapidity,[3]

$$\eta = -\ln\tan\frac{\theta}{2}\,. \tag{1}$$

An observer travelling with the proton sees all rapidities shifted by a constant amount compared to a lab-frame observer. This implies that if he or she defines a distance measure in terms of differences in rapidity, then that distance measure will give the same result independently of the longitudinal frame choice — this is the meaning of 'longitudinally invariant.' Note that the region $\theta = 0 \ldots \pi$ stretches out to become $\eta = \infty \ldots -\infty$, i.e. whereas θ, ϕ describe a point on the surface of a sphere, η, ϕ describe a point on the surface of a cylinder.

Given these kinematic variables, the longitudinally invariant k_t jet finder [10] is essentially an agglomerative hierarchical clustering procedure [2] with a distance measure

$$d_{ij} = \min(k_{ti}^2, k_{tj}^2) R_{ij}^2\,, \qquad R_{ij}^2 = (\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2\,, \tag{2}$$

which is a product of a momentum and a geometrical distance on the surface of a cylinder, R_{ij}. Very roughly speaking this specific form is motivated by the fact that long distance scales (relating to particles that should be clustered first when working backwards through the QCD branching) can correspond either to emissions with small transverse momenta (k_t) or emissions close in angle to a quark or gluon. The detailed definition of the jet finder is given as algorithm 1. Note that in addition to the inter-particle distance d_{ij}, the procedure also makes use of a particle-beam distance $d_{iB} \equiv k_{ti}^2$, and the clustering that

[3] More generally, for massive particles, one uses a definition of rapidity $\eta = \frac{1}{2}\log\frac{E+k_z c}{E-k_z c}$.

occurs depends on whether it is a d_{ij} or a d_{iB} that is smallest. When a d_{iB} the smallest distance, then all $j \neq i$ have $R_{ij} > 1$.

Algorithm 1 Generic definition of the longitudinally invariant k_t jet finder

1: **repeat**
2: For each pair of particles i, j work out the so-called longitudinally invariant k_t distance $d_{ij} = \min(k_{ti}^2, k_{tj}^2) R_{ij}^2$ with $R_{ij}^2 = (\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2$, where k_{ti}, η_i and ϕ_i are the transverse momentum, rapidity and azimuth of particle i; for each parton i also work out the beam distance $d_{iB} = k_{ti}^2$.
3: Find the minimum $d_{\min}$ of all the d_{ij}, d_{iB}.
4: **if** $d_{\min}$ is a d_{ij} **then**
5: Merge particles i and j into a new single particle ℓ, summing their 'four-momenta': $E_\ell = E_i + E_j$, $\vec{k}_\ell = \vec{k}_i + \vec{k}_j$ (alternative recombination schemes are possible).
6: **else** [$d_{\min}$ is a d_{iB}]
7: Declare particle i to be a final-state jet and remove it from the list.
8: **end if**
9: **until** No particles remain.

There exist extensions of the basic procedure of algorithm 1, (a) where d_{ij} is rescaled relative to d_{iB} by a user-chosen factor $1/R^2 \sim 1$ or (b) where clustering is stopped when all d_{ij}, d_{iB} are above a jet resolution threshold d_{cut} (as with usual agglomerative clustering). We here consider only the simplest version, but the arguments below are identical for the general case.

2.2 Brute force and beyond

Algorithm 2 Standard brute-force implementation of k_t clustering

1: Given the initial set of particles, construct a table of all the d_{ij}, d_{iB}. [$\mathcal{O}(N^2)$ operations]
2: **repeat** [Will be done N times]
3: Scan the table to find the minimal value $d_{\min}$ of the d_{ij}, d_{iB}. [$\mathcal{O}(N^2)$ operations]
4: Merge or remove the particles corresponding to $d_{\min}$ as appropriate.
5: Update the table of d_{ij}, d_{iB} to take into account the merging or removal. [$\mathcal{O}(N)$ operations]
6: **until** No particles remain.

The standard reference implementations of the k_t jet finder use a brute-force approach, algorithm 2 [17]. Step 3 dominates, requiring $\mathcal{O}(N^2 \times N = N^3)$ operations and there is an $\mathcal{O}(N^2)$ storage requirement for the table of d_{ij}. One obvious way of improving the speed would be to store the d_{ij} in a balanced binary tree (or some other structure that can act as a priority queue), so that finding the minimum of the d_{ij} at each iteration would have an $\mathcal{O}(\log N)$ cost. There would then be two limiting steps: the construction of the tree would

involve $N^2 \log N$ operations; furthermore after each merging or removal operation, $\mathcal{O}(N)$ of the d_{ij}'s change and accounting for this in the tree structure would involve $\mathcal{O}(N \log N)$ steps at each iteration. Therefore such an algorithm would scale as $N^2 \log N$. Tests using STL priority queues indicate that, in practice, for N of the order of a few thousand, this is no faster than the brute-force N^3 approach.

Since agglomerative clustering with non-geometrical measures is a procedure that is used in a range of fields, a certain amount of work has been carried out to improve on the N^3 algorithm. The earliest dates to Anderberg in 1973 [1] which uses a "nearest neighbour heuristic" (NNH) in which for each i one stores the index j of the closest d_{ij} and then updates this as particles are added and removed. The initialisation for N particles takes $\mathcal{O}(N^2)$ time and adding a particle takes $\mathcal{O}(N)$ time. Removal is more delicate and takes $\mathcal{O}(kN)$ time where k is the number of particles that had the removed particle as their nearest neighbour. In good cases $k \sim 1$ giving an $\mathcal{O}(N^2)$ algorithm, however in the worst case $k \sim N$ leading to an $\mathcal{O}(N^3)$ algorithm.

It turns out that the combination of the distance measure eq. (2) and the structure of radiation in QCD leads to k often being quite high.[4] Therefore the nearest neighbour heuristic, though it improves significantly over the brute force approach, does not in practice behave as an N^2 algorithm. This drawback of the nearest neighbour heuristic is known and a number of approaches for addressing it have been presented in [14, 42], working in $\mathcal{O}(N^2)$ time with $\mathcal{O}(N^2)$ storage, or $\mathcal{O}(N^2 \log N)$ time with $\mathcal{O}(N)$. In our initial investigations we also examined two improvements on the NNH.

Rather than storing just the closest particle to each i, one can introduce a structure of size $N^{1/2}$ for each i, which indicates the nearest neighbour among groups of particles numbered $1 \ldots N^{1/2}$, $(N^{1/2}+1) \ldots 2N^{1/2}$ and so forth. Though initialisation of this structure for all particles takes $\mathcal{O}(N^2)$ time, as for the NNH, recalculating the nearest neighbour of a particle that has lost its current nearest neighbour takes $\mathcal{O}(N^{1/2})$ time. So overall, the algorithm has a worst case clustering time of $\mathcal{O}(N^{5/2})$, while in good cases the time remains $\mathcal{O}(N^2)$, with $\mathcal{O}(N^{3/2})$ storage. This worst and good case time scaling and the need for significant storage are reminiscent of the SuperConga approach of [14].

The second improvement of the NNH is specific to jet clustering and based on the observation that only particles j with $R_{ij} \leq 1$ can lead to a $d_{ij} < d_{iB}$. Therefore one can 'bucket' particles into a tiling of the cylinder based on η, ϕ rectangles with side ≥ 1 and only calculate the d_{ij} for particles j in the same bucket (tile) as i or in one of the 8 surrounding tiles.

The above two improvements lead to run times that are much faster than with the

[4]Specifically the distance measure allows the following worst case: all η_i are identical, the $\Delta\phi$ are 'strongly' ordered, $1 \gg \Delta\phi_{N,N-1} \gg \Delta\phi_{N-1,N-2} \ldots \gg \Delta\phi_{2,1}$ (implying $\Delta\phi_{N,N-1} \simeq \Delta\phi_{N,N-2} \simeq \ldots \simeq \Delta\phi_{N,1}$), and the k_t are even more strongly ordered $k_{t,N} \ll k_{t,N-1} \ll \ldots k_{t,1}$ such that $k_{t,i}\Delta\phi_{i,j} < k_{t,i}\Delta\phi_{i,j-1} \ll k_{t,i-1}\Delta\phi_{i-1,j-1}$ for all $2 < j < i \leq N$. In such a situation the smallest d_{ij} is $d_{N,N-1}$ but at the same time d_{Nj} is smaller than all d_{ij} for $i \neq N$, so the removal of N implies that all particles need to have their nearest neighbour updated. Since QCD radiation tends to produce hierarchies of particles in angles and transverse momenta (though not necessarily both in the order given just above) typically one finds moderately large subsets of particles that fall into the worst case scenario.

brute-force algorithm, and also a significant improvement relative to the NNH. However they remain worse than those that can be obtained with N^2 algorithms based on the geometric separation to be discussed below, therefore we shall not elaborate on them any further (nor have we investigated the full range of alternative algorithms discussed in [14, 42] — from the relative timing results shown there, we suspect them to be similar to ours, bearing in mind that our tiling improvement does not have a direct counterpart in the more general algorithms).

3 Geometrical algorithms

We observe that given the specific form of the distance measure, a product of a momentum scale and a geometrical distance, the following statement can be made:

Lemma: If i, j form the smallest d_{ij}, and $k_{ti} < k_{tj}$, then $R_{ij} < R_{i\ell}$ for all $\ell \neq j$, i.e. j is the (cylindrical) geometrical nearest neighbour of particle i.

Proof: Suppose the Lemma is wrong and that there exists a particle ℓ such that $R_{i\ell} \leq R_{ij}$: then $d_{i\ell} = \min(k_{ti}^2, k_{t\ell}^2) R_{i\ell}^2$ and since $\min(k_{ti}^2, k_{t\ell}^2) \leq k_{ti}^2$, we have that $d_{i\ell} \leq d_{ij}$, in contradiction with the statement that i and j have the smallest d_{ij}.

Given this observation, one can formulate the k_t jet finding problem in geometrical terms, algorithm 3.

Algorithm 3 Geometrical formulation of the k_t jet finder

1: For each particle i establish its cylindrical geometrical nearest neighbour $\mathcal{C}_i$ and calculate the $d_i = \min(d_{i\mathcal{C}_i}, d_{iB})$.
2: **repeat**
3: Find the minimal value $d_{\min}$ of the d_i.
4: Merge or remove the particles corresponding to $d_{\min}$, as appropriate.
5: If a new particle ℓ has been created determine its cylindrical geometrical nearest neighbour $\mathcal{C}_i$.
6: Determine which other particles' nearest neighbours have changed as the result of the removal of i and j and addition of ℓ.
7: Determine the d_i for each particle whose nearest neighbour has changed and, where relevant, for the new particle ℓ.
8: **until** No particles remain.

The crucial improvement here is that $d_{\min}$ no longer has to be searched for among $\mathcal{O}(N^2)$ d_{ij} entries, but only among $\mathcal{O}(N)$ d_i. The problem then has two separate components: that of finding the minimum of the d_i and that of establishing and keeping track of all nearest neighbours (or equivalently, that of maintaining the nearest neighbour graph).

3.1 N^2 formulations

A formulation that has a worst-case $\mathcal{O}(N^2)$ behaviour can be obtained based on the following observations. Finding the minimum of $\mathcal{O}(N)$ d_i values, N times is overall $\mathcal{O}(N^2)$. The initial set of geometrical cylindrical nearest neighbours (CNN) can be established in time N^2, and because no point can be the CNN of more than $\mathcal{O}(1)$ others (as discussed below the cylindrical problem is similar to a planar one, where the limit is 6 [30]) we only need to recalculate $\mathcal{O}(N)$ distances to reestablish the CNN information after an update. Specifically when deleting i and j and inserting ℓ, one needs to scan all N points to see which ones were the CNN of either i and j (their CNNs should then be recalculated) and which ones have acquired ℓ as their CNN or become the CNN of ℓ. Repetition over N iterations gives an $\mathcal{O}(N^2)$ behaviour. We call this the *naive* N^2 algorithm.

Following the discussion in section 2.2, we do not need to search for CNNs on the whole cylinder, because if $R_{ij} > 1$ then $d_{ij} > d_{iB}$ and there will never be a recombination between i and j. One can therefore tile the cylinder, as in section 2.2, and for a given point, limit the search for CNNs to its own tile and the surrounding tiles. Asymptotically, the algorithm will remain N^2, both because of the search for $d_{\min}$ (though this is easily improved) and because in practice, for large N the density of points on the cylinder is proportional to N (and therefore so is the number of points in a tile). This will be called the *tiled* N^2 algorithm.

3.2 $N \log N$ formulation

Let us first reexamine the cost of finding the minimum of the d_i: in step 1 one can create a priority queue containing all the d_i in time $\mathcal{O}(N \log N)$. In step 3 finding the minimal entry in the priority queue will take at most time $\mathcal{O}(\log N)$. In step 7 (in the worst case of there being a recombination) one needs to update the queue to account for the removal of the particles that recombined, i and j, the addition of the new particle ℓ and also for all particles whose CNN has changed due to the removal of i and j and the addition of ℓ. As discussed just above, the number of points that can be the CNN of i, j or ℓ is $\mathcal{O}(1)$, so updating the priority queue will take $\mathcal{O}(\log N)$ time. Since steps 3 and 7 are to be repeated N times, the overall cost of the handling of the d_i priority queue is $\mathcal{O}(N \log N)$.

The geometrical nearest-neighbour graph part of the problem takes place on the surface of a cylinder, since ϕ is periodic in 2π. It is straightforward to reduce this to a planar problem, for which far more results are known, for example by unrolling the cylinder and then making copies of points displaced by 2π so as to account for the proximity of points near $\phi = 0$ and $\phi = 2\pi$.[5] The overhead of making the copies is at most a factor of order 1 (and in practice, for uniformly distributed points, can be reduced to $1 + \mathcal{O}(1/\sqrt{N})$).

The problem of searching for nearest neighbours on the plane has been extensively

[5]This procedure will misbehave if $R_{i\mathcal{C}_i} > 2\pi$, since i's copy, i', will be identified as a closer neighbour of i at distance 2π. However, as we have already noted a couple of times, if $R_{i\mathcal{C}_i} > 1$ then $d_{i\mathcal{C}_i} > d_{iB}$ and $d_i = d_{iB}$, so that we do not actually need to know $\mathcal{C}_i$. In a sense, it is not the full nearest-neighbour graph that interests us, but only a range-bounded nearest-neighbour graph.

studied (in what follows, nearest neighbours will now always be meant with respect to the planar geometrical distance, rather than d_{ij} or the cylindrical R_{ij}). Various structures are known for finding the nearest neighbours of all points on a plane, among them k-d trees [7] and Delaunay triangulations (or their duals, Voronoi diagrams) [5]. Several codes and algorithms exist for the creation both of static k-d trees (e.g. [35]) and static Voronoi/Delaunay diagrams (e.g. [26, 38]) in $N \log N$ time. The determination of all nearest neighbours takes at most a further time $N \log N$ ($\mathcal{O}(N)$ in the Delaunay case).

To maintain all necessary nearest neighbour information across the updates that will occur during the clustering, additional features are required. Firstly, it should be possible to both add and delete points in the structure. Secondly, in addition to being able to find the nearest neighbour of a point in the structure, one needs to be able to carry out *reverse* nearest neighbour queries [28], i.e. establish for a given point i, which other points in the structure have i as their nearest neighbour. As pointed out in [28] this is sufficient in order to maintain the nearest-neighbour graph.

The problem of maintaining dynamic structures that can answer such queries and be updated efficiently (in $\mathcal{O}(\log N)$ time) is a current one [46]. Given that the nearest-neighbour graph is a subgraph of the Delaunay Triangulation (DT), one of the simplest approaches to maintaining the nearest neighbour graph is to make use of methods that allow efficient dynamic updates of the DT, in particular those allowing for both insertion and deletion [12, 11, 29, 32, 23, 24].[6] The analyses of these structures involve some form of assumption of randomisation, for example that one inserts points in random order and that when deleting a point it is chosen at random from the point set. With such an assumption it is possible to achieve average update times of $\mathcal{O}(\log N)$. Recently an approach has been developed that is more robust with respect to the randomness of insertions and deletions [22] (see also [21]) however the maintenance of the nearest-neighbour graph takes a time $\log^6 N$ per update.

Of the above approaches, that of [24], the DT hierarchy, has been incorporated into the triangulations component [9] of CGAL [15] and is therefore readily accessible. This motivated us to choose the CGAL package for the computational geometry part of our jet clustering application.

Within CGAL, initial nearest-neighbour information can be set up for each vertex i by circulating over all of incident vertices in the DT and selecting the one that is closest. When removing a point i, one should carry out a reverse nearest neighbour query, by circulating over the incident vertices of i and establishing the subset $\{\mathrm{RNN}_i\}$ that had i as their nearest neighbour. After removal of i one should establish the new nearest neighbour of each of the elements $\{\mathrm{RNN}_i\}$ (by circulating over their incident vertices in the DT). Finally, when a point i is added one should circulate over its incident vertices and establish i's nearest neighbour. In the process one should verify whether i is closer to any of incident vertices than their currently recorded nearest neighbours and if so, rectify that information.

[6] There has also been work in the context of R-trees [27, 31] that does satisfy all our requirements (notably [33, 45]), however with R-trees one of the main aims is to achieve efficiency when data is kept on disk, which is not a concern in our case. In practice it seems that the CPU part of documented running times [45] does not compare favourably with the results we shall show in section 4).

Given that the total number of vertices in the DT is bounded $\mathcal{O}(N)$, initially establishing the nearest-neighbour information takes $\mathcal{O}(N)$ time. Reestablishing the nearest-neighbour information takes a time $\mathcal{O}(k)$, with k, expected $\mathcal{O}(1)$, the degree of the vertices around which one circulates in the DT. Therefore keeping track of nearest neighbour information will take an overall time $\mathcal{O}(N)$, whereas creating and maintaining the DT hierarchy across $\mathcal{O}(N)$ updates is expected $\mathcal{O}(N \log N)$.

There are caveats related to the meaning of 'expected'.[7] The jet clustering procedure removes and inserts points deterministically and one might worry about the existence of pathological sets of input particle momenta that bring out the worst case of the DT hierarchy updates. In particular it is known that the time for deletion is actually $\mathcal{O}(k \log k)$ [23], where k is the degree of the vertex being deleted. Large k ($\sim N$), corresponds to the situation where one point is at the centre of a circle and all others on its circumference. This is extremely unlikely among the configurations that will occur in particle physics, where the distribution of k should not be too different from that in a random set of points. Furthermore it would only be dangerous if the algorithm were then to repeatedly remove and reinsert the central point, leading to an overall $N^2 \log N$ cost. Detailed analysis of the clustering procedure suggests that this cannot occur.[8] Another potential problem with high k relates to the maintenance of the nearest neighbour information. Even if the central point is not directly involved in a recombination, it may have had as nearest neighbour a point that was involved (and was removed). In such a case the central point's nearest neighbour has to be redetermined at cost $\mathcal{O}(k)$. One can devise extremely unlikely pathological configurations[9] where this will occur repeatedly across a sequence of recombinations of length $\mathcal{O}(k)$, in which case if $k \sim N$ the overall cost will be $\mathcal{O}(N^2)$.

These special cases aside, let us summarise the overall computation complexity of the algorithm: the total time spent in setting up CGAL's Delaunay Triangulation hierarchy is expected $\mathcal{O}(N \log N)$ and a further expected time $\mathcal{O}(N \log N)$ is spent updating and extracting information from it during the iteration of the jet-finding procedure. Maintaining the $\mathcal{O}(N)$ d_i entries in a priority queue for easy minimum searching also takes $\mathcal{O}(N \log N)$ time (in practice we actually use a balanced binary tree), so the overall time for running the jet finding is expected $\mathcal{O}(N \log N)$.

[7] We wish to thank O. Devillers for correspondence on this point.

[8] For the point i at the centre to be removed it must form a d_{min} with a point j on the circumference. Two scenarios exist: $\mathcal{C}_j = i$ or $\mathcal{C}_i = j$. The former is impossible in the limit of there being many points on the circumference (since $\mathcal{C}_j$ will be another point on the circumference). Regarding the latter, d_{ij} can be a $d_{\min}$ if $k_{ti}^2 R_{ij}^2 < k_{tj}^2 R_{j\mathcal{C}_j}^2$, however since there are many points on the circumference $R_{j\mathcal{C}_j}^2 \ll R_{ij}^2$, implying $k_{ti}^2 \ll k_{tj}^2$. In such a case, if i and j recombine, then because $k_{ti} \ll k_{tj}$ the new particle to be inserted will be close to the original position of j (the position is approximately given by a k_t-weighted average), and therefore not in the centre of the circle.

[9] Let momentum k_1 be the central point, with momenta $k_2 \ldots k_n$ distributed in a spiral around k_1 such that $R_{1,i+1} < R_{1,i}$ and $k_{t,i+1} \ll k_{t,i}$. The nearest neighbour of the central point is k_n and the first recombination will be between k_n and k_{n-1}. Removing k_n implies a redetermination of k_1's nearest neighbour. This will turn out to be the recombined particle, which will have taken k_{n-1}'s location. Thus the clustering procedure will work its way around the spiral, recomputing the central point's nearest neighbour at each step.

4 Timing measurements

While the CGAL-based algorithm is clearly expected to be the best for asymptotically large N, it is necessary to examine how it fares for the values of N that are relevant at high-energy colliders. A typical proton-proton collision produces a few hundred particles. Two factors can lead to events having far more particles however. Firstly, in order to be able to search for potentially very rare signatures of new particles, rather than having a single collision at a time, the LHC will have about 20 collisions (proton-proton interactions) occurring simultaneously, leading to a few thousand particles being produced per event; and in a lead-lead ('heavy-ion') collision, there will be so many interactions between the individual protons in the lead nuclei, that many tens of thousands of particles are expected to be produced [44]. The second aspect is that some parts of the particle detectors are segmented in a fine grid-like structure (in η and ϕ) and it can be simplest experimentally (e.g. due to noise-related issues) to run the jet-finding as if each element of the grid represented a particle in its own right. The number of grid elements in a typical detector ranges from 10^3 to 10^4 [39]. Therefore the overall range of N that needs to be studied is 10^2 to 10^5.

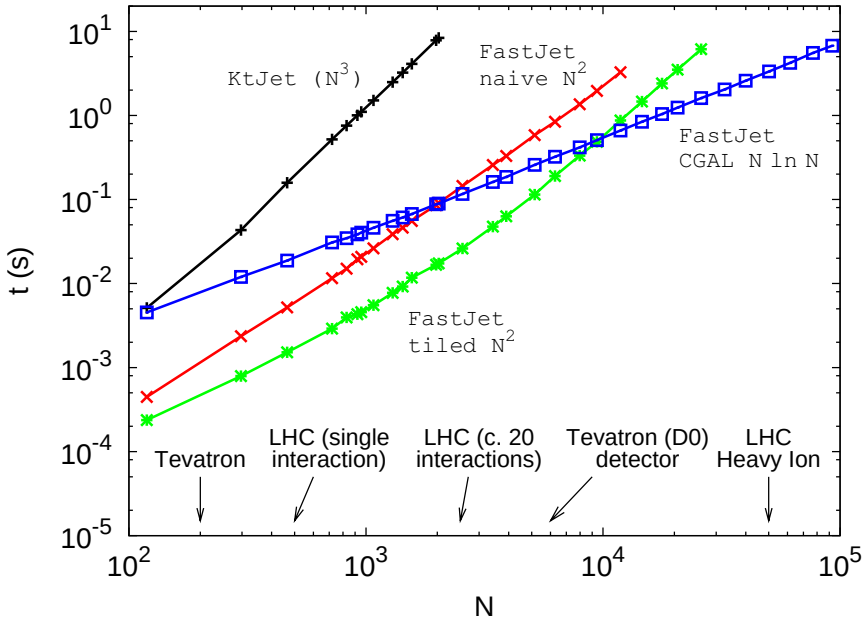

Figure 2: Timings of various algorithms for the k_t jet-finder as a function of the number of particles N, together with indications of the contexts in which various values of N can arise. The events used for the jet finding were generated with the Pythia Monte Carlo event generator [18]. Measurements were performed on a Pentium IV processor running at 3.06 GHz with 1GB of memory.

The results for the timings might depend significantly on the structure of the events being used, in particular if one deliberately includes worst-case events. To cover a large range of N in a representative manner, we used the `PYTHIA` Monte Carlo program [18] to

generate simulated proton-proton collision events. These simulations are in part based on theoretical calculations, and in part on fits to real data. We build events by taking one simulated event with pronounced jets (consisting of about 100 particles) and superimposing varying numbers of simulated typical proton-proton collisions, known as 'minimum-bias' events. These do not usually have jets and are a form of background that will be present when multiple proton-proton collisions occur simultaneously.

Figure 2 shows timings for implementations of the k_t jet finder with various algorithms: `KtJet` [17] codes the N^3 algorithm described in section 2. The `FastJet` algorithms are those based on the geometrical approach of section 3 in the implementation available from [48]. One sees that the geometrical algorithms all run substantially faster than the original N^3 algorithm and their N-dependences coincide approximately with the expected scalings. The CGAL based $N \log N$ algorithm becomes better than the naive N^2 algorithm for $N \gtrsim 2000$ and better than the tiled N^2 algorithm for $N \gtrsim 10^4$. Systematically taking the best of the various algorithms brings overall computation time for dealing with 10^9 LHC proton-proton events or for 10^7 heavy-ion events into the ballpark of a few months of CPU time. In the heavy-ion case, CGAL allows an improvement over the tiled N^2 algorithm by about an order of magnitude.

The above timings are eminently manageable by LHC standards. In contrast, with the original brute force N^3 algorithm, processing times would have been in the range $10^2 - 10^4$ CPU years, which would have represented a non-negligible burden even given the extensive grid computing facilities that will be available to the LHC experiments.[10]

Concentrating specifically on the $N \log N$ CGAL-based algorithm it is interesting to examine which parts are the most time consuming. Considering $N = 10^4$, the two $N \log N$ parts are the priority queue, which consumes about 5% of the time and CGAL which takes about 65%. The latter breaks down into about 30% each for insertion and deletion and 5% for circulation. Of the remaining 30%, about half is directly associated with maintenance of the nearest neighbour information. One notes that for this value of N, parts expected to scale as $N \log N$ (insertion and the priority queue) take about the same amount of time as parts that scale as N.

Given that the $N \log N$ behaviour is only an expectation, based on the assumption of insertion and deletion being sufficiently random, it is interesting to verify whether the measured timings do actually follow this expectation. To this end, figure 3 shows timings divided by N for the CGAL-based algorithm — the result should be linear in $\log N$. The two event samples used here extend to much larger N than that of fig. 2, so as to provide a clear view of the asymptotic region. One event sample superimposes multiple minimum bias events (as in fig. 2), while the other adds particles on a grid that is made progressively finer towards large N, mimicking the structure of a detector. In both cases there is strong evidence for $N \log N$ behaviour, though the specific structure of the events does affect the coefficient of $N \log N$. Thus, despite general worries that the non-random nature of a

[10] One should bear in mind also that it will be interesting to the run the jet-finder more than once, for example on real data and multiple simulated datasets, and also examining the variants mentioned in section 2

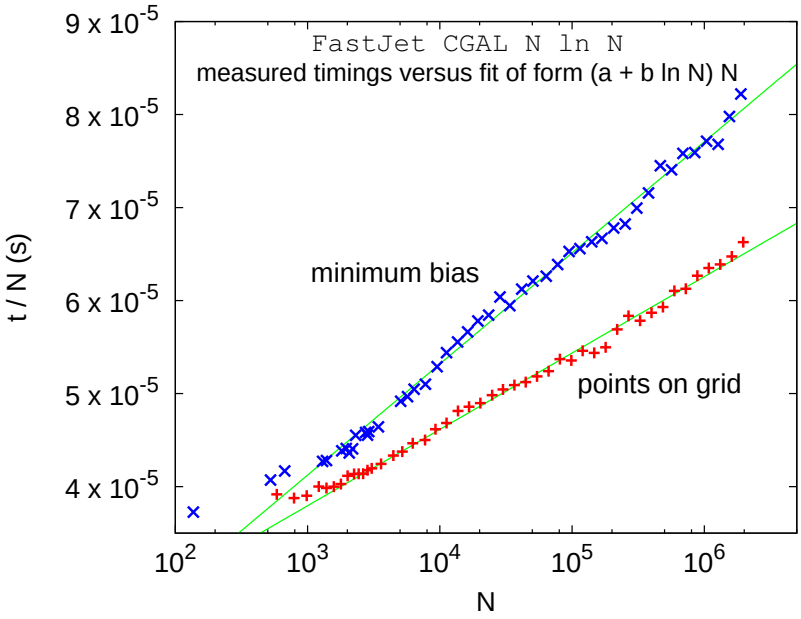

Figure 3: Verification of the $N \log N$ behaviour of the CGAL based algorithm. The points are measured timings t (divided by N), while the lines correspond to fits of the form $t = (a + b \log N)N$ for $N > 3000$. Two simulated sets of momenta have been used for these tests, both described in the text.

procedure such as clustering might be fatal to the assumptions behind the expectation of $N \log N$ timing (as expressed for example in [22]), for jet clustering this does not seem to be an issue.

5 Conclusions

Jet-finding in high-energy particle collisions with the widely advocated longitudinally invariant k_t agglomerative clustering jet finder is highly CPU intensive with the brute force N^3 approach used up to now. Our geometrical reformulation of the problem, in terms of a planar nearest neighbour graph, leads to practical improvements of several orders of magnitude in speed. For moderately large N the resulting simple N^2 algorithms are remarkably efficient. For very large N, the methods of computational geometry, in particular CGAL's implementation of the Hierarchical Delaunay Triangulation, make it possible to construct a dynamic planar nearest-neighbour graph with expected $\mathcal{O}(\log N)$ update time. This is the key non-trivial ingredient in reducing the complexity of the problem from $\mathcal{O}(N^2)$ to expected $\mathcal{O}(N \log N)$. The impact of the use of CGAL will be most significant for heavy-ion collisions, where $N \simeq 50000$ and the CGAL $N \log N$ algorithm is an order magnitude faster than all others.

To our knowledge this is the first use of computational geometry in the study of particle physics event properties. We suspect that there may also exist other applications. In

particular to account for the contamination from minimum bias events (a form of noise) it will be important to be able to provide an estimate of the area of a jet and it can be shown that for this purpose the Voronoi diagram has a special correspondence to the underlying physics [41]. Additionally there exist agglomerative clustering jet finders with other distance measures, notably the Cambridge/Aachen variant [13] which has a purely geometrical (cylindrical) measure and so could be dealt with in a manner similar to that described here, or alternatively using dedicated planar closest pair algorithms such as [16, 8].

Acknowledgements

We wish to thank O. Devillers for helpful correspondence and J.M. Butterworth, J. Huston, C. Roland and M. H. Seymour for discussions. This work was supported in part by grant ANR-05-JCJC-0046-01 from the French Agence Nationale de la Recherche.

References

[1] M. R. Anderberg, Cluster Analysis for Applications, (Number 19 in Probability and Mathematical Statistics, Academic Press, New York, 1973).

[2] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, (Wiley-interscience, 2nd edition, 2000).

[3] R. K. Ellis, W. J. Stirling and B. R. Webber QCD and Collider Physics, (Cambridge University Press, 1996).

[4] Donald H. Perkins, Introduction to High Energy Physics (Cambridge University Press, 2000, 2nd edition).

[5] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental data structure, ACM Computing Surveys 23 (1991) 345.

[6] W. Bartel *et al.* [JADE Collaboration], Experimental studies on multi-jet production in e^+e^- annihilation at Petra energies, Z. Phys. C 33 (1986) 23;
S. Bethke *et al.* [JADE Collaboration], Experimental investigation of the energy dependence of the strong coupling strength, Phys. Lett. B 213 (1988) 235.

[7] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (1975) 509.

[8] S. N. Bespamyatnikh, An optimal algorithm for closest-pair maintenance, Discrete Comput Geom 19 (1998) 175.

[9] J.-D. Boissonnat *et al.*, Comp. Geom.: Theory and Applications 22 (2001) 5.

[10] S. Catani, Y. L. Dokshitzer, M. Olsson, G. Turnock and B. R. Webber, New clustering algorithm for multi-jet cross-sections in e+ e- annihilation, Phys. Lett. B 269, 432 (1991);
S. Catani, Y. L. Dokshitzer, M. H. Seymour and B. R. Webber, Longitudinally invariant k_t clustering algorithms for hadron hadron collisions, Nucl. Phys. B 406 (1993) 187;
S. D. Ellis and D. E. Soper, Successive combination jet algorithm for hadron collisions, Phys. Rev. D 48 (1993) 3160 [hep-ph/9305266].

[11] K. L. Clarkson, K. Mehlhorn and R. Seidel, Four results on randomized incremental constructions, Comp. Geom.: Theory and Applications 3 (1993) 185.

[12] O. Devillers, S. Meiser, M. Teillaud, Fully dynamic Delaunay triangulation in logarithmic expected time per operation Comp. Geom.: Theory and Applications 2, 55 (1992).

[13] Y. L. Dokshitzer, G. D. Leder, S. Moretti and B. R. Webber, Better jet clustering algorithms, JHEP 9708 (1997) 001 [hep-ph/9707323];
M. Wobisch, Ph.D. thesis, RWTH Aachen, 1999, DESY-THESIS-2000-049.

[14] D. Eppstein Fast hierarchical clustering and other applications of dynamic closest pairs, J. Experimental Algorithmics 5 (2000) 1-23 [cs.DS/9912014].

[15] A. Fabri *et al.*, On the design of CGAL a computational geometry algorithms library Softw. Pract. Exper. 30 (2000) 1167

[16] M. J. Golin, R. Raman, C. Schwarz, M. H. M. Smid Randomized data structures for the dynamic closest-pair problem SIAM J. Comput. 27 (1998) 1036.

[17] M. H. Seymour, `KtClus`, `http://hepforge.cedar.ac.uk/ktjet/fortran/`;
J. M. Butterworth, J. P. Couchman, B. E. Cox and B. M. Waugh, KtJet: A C++ implementation of the k_t clustering algorithm (`KtJet`), Comput. Phys. Commun. 153, 85 (2003) [hep-ph/0210022].

[18] T. Sjostrand *et al.*, High-energy-physics event generation with PYTHIA 6.1, Comput. Phys. Commun. 135, 238 (2001) [hep-ph/0010017];
T. Sjostrand *et al.*, PYTHIA 6.3: Physics and manual, hep-ph/0308153.

[19] G. Sterman and S. Weinberg, Jets from quantum chromodynamics, Phys. Rev. Lett. 39 (1977) 1436.

[20] F. V. Tkachov, Measuring multijet structure of hadronic energy flow or what is a jet?, Int. J. Mod. Phys. A 12 (1997) 5411 [arXiv:hep-ph/9601308];
D. Y. Grigoriev, E. Jankowski and F. V. Tkachov, Towards a standard jet definition, Phys. Rev. Lett. 91, 061801 (2003) [hep-ph/0301185];
D. Y. Grigoriev, E. Jankowski and F. V. Tkachov, Optimal jet finder, Comput. Phys. Commun. 155, 42 (2003) [hep-ph/0301226].

[21] B. Aronov *et al.*, Data Structures for Halfplane Proximity Queries and Incremental Voronoi Diagrams, in Proc. of the 7th Latin American Symposium on Theoretical Informatics, Valdivia, Chile, March 2006, cs.CG/0512091.

[22] T. M. Chan, A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. in Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms, 2006.

[23] O. Devillers, On Deletion in Delaunay Triangulation, cs.CG/9907023.

[24] O. Devillers, Improved incremental randomized Delaunay triangulation, In Proc. 14th Annu. ACM Sympos. Comput. Geom., p. 106 (1998) [cs.CG/9907024];
O. Devillers, The Delaunay hierarchy, Internat. J. Found. Comput. Sci. 13 (2002) 163.

[25] V. D. Elvira [D0 Collaboration], Jet measurements at D0 using a k_t algorithm, Nucl. Phys. Proc. Suppl. 121 (2003) 21 [hep-ex/0209073].

[26] S. Fortune, A sweepline algorithm for Voronoi diagrams, in Proc. of the second annual symposium on Computational geometry, p. 312 (1986).

[27] A. Guttman, R-Trees: a dynamic index structure for spatial searching, in Proc. 1984 ACM-SIGMOD Conference on Management of Data (1985) 47.

[28] F. Korn and S. Muthukrishnan, Influence sets based on reverse nearest neighbor queries, In Proc. ACM SIGMOD Int. Conf. on Management of Data, Dallas, USA, May 2000.

[29] K. Mulmuley, Randomized multi-dimensional search trees, in Proc. 7th ACM Symposium on Comp. Geom. in North Conway, p. 121 (1991).

[30] M. S. Paterson and F. F. Yao, On nearest-neighbor graphs, in Proc. 19th Internat. Colloq. Automata Lang. Program., volume 623 of Lecture Notes Comput. Sci., p. 416. Springer-Verlag, 1992.

[31] N. Roussopoulos, S. Kelley and F. Vincent, Nearest neighbor queries, in Proc. of ACM Sigmod, May 1995.

[32] O. Schwarzkopf, Dynamic maintenance of geometric structures made easy, in Proc. 32nd IEEE Sympos. Found. Comp. Sci., p. 197 (1991).

[33] I. Stanoi, D. Agrawal, and A. El Abbadi, Reverse nearest neighbor queries for dynamic databases, in Proc. of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD), 2000.

[34] See e.g. J. Womersley, Tests of perturbative QCD and jet physics, Int. J. Mod. Phys. A 15S1, 607 (2000) [eConf C990809 (2000) 607] [hep-ex/9912009].

[35] ANN: A Library for approximate nearest neighbor searching, `http://www.cs.umd.edu/~mount/ANN/`.

[36] ATLAS collaboration, Detector and physics performance Technical Design Report, http://cern.ch/Atlas/GROUPS/PHYSICS/TDR/access.html.

[37] A. Banfi, G. P. Salam and G. Zanderighi, Infrared safe definition of jet flavour, hep-ph/0601139.

[38] B. Barber, Qhull, http://www.qhull.org/

[39] G. C. Blazey *et al.*, Run II jet physics, hep-ex/0005012.

[40] M. Cacciari and G. P. Salam, Dispelling the N^3 myth for the k_t jet-finder, hep-ph/0512210.

[41] M. Cacciari and G. P. Salam, in preparation.

[42] J. Cardinal and D. Eppstein, Lazy algorithms for dynamic closest pair with arbitrary distance measures, Tech. Rep. 502, Univ. Libre de Bruxelles, 2003.

[43] A. Abulencia *et al.* [CDF II Collaboration], Measurement of the inclusive jet cross section using the k_t algorithm in p anti-p collisions at $\sqrt{s} = 1.96\,\mathrm{TeV}$, hep-ex/0512062.

[44] C. A. Loizides, Jet physics in ALICE, Ph.D. thesis, Frankfurt University, nucl-ex/0501017.

[45] Y. Tao, D. Papadias and X. Lian, Reverse kNN Search in arbitrary dimensionality (2004).

[46] The dynamic nearest neighbour problem on the plane is described as being unsolved at http://maven.smith.edu/~orourke/TOPP/P63.html#Problem.63

[47] For an extended gentle introduction to the basics of particle physics, see http://particleadventure.org/.

[48] The FastJet program is available from http://www.lpthe.jussieu.fr/~salam/fastjet/.