Dispelling the N^3 myth for the k_t jet-finder

Matteo Cacciari, Gavin P. Salam [*]

LPTHE, Universities of Paris VI and VII and CNRS, Paris, France

Abstract

At high-energy colliders, jets of hadrons are the observable counterparts of the perturbative concepts of quarks and gluons. Good procedures for identifying jets are central to experimental analyses and comparisons with theory. The k_t family of successive recombination jet finders has been widely advocated because of its conceptual simplicity and flexibility and its unique ability to approximately reconstruct the partonic branching sequence in an event. Until now however, it had been believed that for an ensemble of N particles the algorithmic complexity of the k_t jet finder scaled as N^3, a severe issue in the high multiplicity environments of LHC and heavy-ion colliders. We here show that the computationally complex part of k_t jet-clustering can be reduced to two-dimensional nearest neighbour location for a dynamic set of points. Borrowing techniques developed for this extensively studied problem in computational geometry, k_t jet-finding can then be performed in $N \ln N$ time. Code based on these ideas is found to run faster than all other jet finders in current use.

© 2006 Elsevier B.V. Open access under CC BY license.

1. Introduction

Partons (quarks and gluons), are the concepts that are central to discussions of the QCD aspects of high-energy collisions such as those at the Fermilab Tevatron and the future Large Hadron Collider (LHC) at CERN. Quarks and gluons, however, are not observable, and in their place one sees *jets*, collimated bunches of high-energy hadrons which are the result of the fragmentation and hadronisation of the original hard (high-energy) partons. Today's limited understanding of non-perturbative QCD is such that it is not currently possible to predict the exact patterns of hadrons produced. Instead one makes predictions in terms of quarks and gluons and relates these to observations in terms of hadron jets.

Naively, jets are easily identified—one simply searches for bunches of collimated hadrons. However, to carry out accurate comparisons between parton-level predictions and hadron-level observations one needs a well-defined 'jet-finding' procedure. The jet-finder is applied both to perturbatively predicted partonic configurations and to observed hadronic configurations and one then directly compares distributions for the predicted partonic jets and the observed hadronic jets. Though partonic and hadronic jets are not equivalent, there is strong evidence (theoretical [1] and experimental [2]) that the comparison can be performed with controlled accuracy.

Insofar as jet-finding is an approximate attempt to invert the quantum mechanical processes of QCD branching and hadronisation, it is not a unique procedure. Various kinds of jet-finders have been proposed, among them cone-type [1,3] and sequential-clustering [4–7] jet-finders (for alternatives, see [8–11]).

Cone jet-finders are the most frequently used at the Tevatron. They are based on identifying energy-flow into cones in (pseudo)rapidity $\eta = -\ln\tan\theta/2$ and azimuth ϕ, together with various steps of iteration, merging and splitting of the cones to obtain the final jets. Cone jet-finders tend to be rather complex, different experiments have used different variants (some of them infrared unsafe), and it is often difficult to know exactly which jet-finder to use in theoretical comparisons.

In contrast, the cluster-type jet-finders, generally based on successive pair-wise recombination of particles, have simple definitions and are all infrared safe (for reviews see [12,13]). We shall focus here on the most widely used of them, the k_t jet-finder [5], defined below. Among its physics advantages are (a)

[*] Corresponding author.
E-mail address: salam@lpthe.jussieu.fr (G.P. Salam).

that it purposely mimics a walk backwards through the QCD branching sequence, which means that reconstructed jets naturally collect most of the particles radiated from an original hard parton, giving better particle mass measurements [14,15], general kinematic reconstruction [16] and gaps-between-jets identification [17] (of relevance to Higgs searches); and (b) it allows one to decompose a jet into constituent subjets, which is useful for identifying decay products of fast-moving heavy particles (see e.g. [18]) and various QCD studies. This has led to the widespread adoption of the k_t jet-finder in the LEP (e^+e^- collisions) and HERA (ep) communities.

Despite its advantages, k_t clustering has so far seen only limited study [19–21] at the Tevatron. The reasons for this are not entirely clear. One known drawback of the k_t jet finder for high-multiplicity hadron-collider environments is its apparent algorithmic slowness: to cluster N particles into jets requires $\mathcal{O}(N^3)$ operations in current implementations [22]. For a typical event at the upcoming LHC, with an expected multiplicity of $N = \mathcal{O}(2000)$, this translates into a clustering time of $\mathcal{O}(10\,\mathrm{s})$ of CPU time on a modern $\mathcal{O}(3\,\mathrm{GHz})$ processor; this is considerable given that the clustering has to be repeated for millions of events. For a typical heavy-ion event at LHC, where $N = \mathcal{O}(50\,000)$, the clustering time would grow to an unsustainable $\mathcal{O}(10^5\,\mathrm{s})$, i.e. more than one day! Even at the Tevatron, where the multiplicity is quite modest, the fact that noise may cause the number of active calorimeter cells to be far larger than the number of particles has led to the use of a complex (and physically questionable) preclustering procedure prior to running the k_t jet finder, so as to reduce the effective value of N to something that is manageable [20].

The slowness of the k_t jet-finder has been one of the motivating factors behind proposals for alternative jet-finders [9,10]. Here we will show that the k_t jet-finder can in fact be formulated in an algorithmically fast ($N \ln N$) manner. A C++ implementation of this (and a related N^2) algorithm[1] will be shown to run orders of magnitude faster than currently available implementations, making it feasible (and easy) to use the k_t jet finder for efficiently studying high-multiplicity events.

2. The k_t jet-finder

The k_t jet finder, in the longitudinally invariant formulation suitable for hadron colliders, is defined as follows:

(1) For each pair of particles i, j work out the 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 $d_{iB} = k_{ti}^2$.
(2) Find the minimum d_{min} of all the d_{ij}, d_{iB}. If d_{min} is a d_{ij} merge particles i and j into a single particle, summing their four-momenta (alternative recombination schemes are pos-

sible); if it is a d_{iB} then declare particle i to be a final jet and remove it from the list.
(3) Repeat from step 1 until no particles are left.

There exist extensions of this basic procedure, (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}. We here consider only the simplest version, as given above, but the arguments below are identical for the general case.

Now we reconsider the above procedure, making explicit the computational overheads of the various steps as implemented in standard jet finding codes [22].

(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}.
     [$\mathcal{O}(N^2)$ *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.*]

Step (2) dominates, requiring $\mathcal{O}(N^2 \times N = N^3)$ operations.[2]

3. The `FastJet` algorithm

To obtain a better algorithm we isolate the geometrical aspects of the problem, with the help of the following observation.

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 geometrical nearest neighbour of particle i.*

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

This means that if we can identify each particle's geometrical nearest neighbour (in terms of the geometrical R_{ij} distance), then we need not construct a size-N^2 table of $d_{ij} = \min(k_{ti}^2, k_{tj}^2)R_{ij}^2$, but only the size-$N$ array, $d_{i\mathcal{G}_i}$, where $\mathcal{G}_i$ is i's geometrical nearest neighbour.[3] We can therefore write the following algorithm:

---

[1] 'Jet-algorithm' is often used in the literature to refer to the choice of the rules for finding a jet; here instead 'algorithm' refers to the translation of a given set of jet-finding rules into explicit steps on a computer.

[2] One notes also the storage requirement in step (1) of $4N^2 + \mathcal{O}(N)$ bytes (double precision), which is manageable for $N = 1000$ but becomes an issue in heavy-ion environments with up to $50\,000$ particles. At the (substantial) expense of recalculating the $\mathcal{O}(N^2)$ d_{ij} at each iteration, the storage issue can be eliminated.

[3] We shall drop 'geometrical' in the following, speaking simply of a 'nearest neighbour'.

(1) For each particle i establish its nearest neighbour $\mathcal{G}_i$ and construct the arrays of the $d_{i\mathcal{G}_i}$ and d_{iB}.
(2) Find the minimal value $d_{\min}$ of the $d_{i\mathcal{G}_i}$, d_{iB}.
(3) Merge or remove the particles corresponding to $d_{\min}$ as appropriate.
(4) Identify which particles' nearest neighbours have changed and update the arrays of $d_{i\mathcal{G}_i}$ and d_{iB}. If any particles are left go to step (2).

This already reduces the problem to one of complexity N^2: for each particle we can find its nearest neighbour by scanning through all $\mathcal{O}(N)$ other particles [$\mathcal{O}(N^2)$ operations]; calculating the $d_{i\mathcal{G}_i}$, d_{iB} requires $\mathcal{O}(N)$ operations; scanning through the $d_{i\mathcal{G}_i}$, d_{iB} to find the minimal value $d_{\min}$ takes $\mathcal{O}(N)$ operations [to be repeated N times]; and after a merging or removal, updating the nearest neighbour information will require $\mathcal{O}(N)$ operations [to be repeated N times].[4]

We note, though, that three steps of this algorithm—initial nearest neighbour identification, finding $d_{\min}$ at each iteration, and updating the nearest neighbour information at each iteration—bear close resemblance to problems studied in the computer science literature and for which efficient solutions are known:

- Given an ensemble of vertices in a plane (specified by the η_i and ϕ_i of the particles), to find the nearest neighbour of each vertex one can use a structure known as a Voronoi diagram [23] or its dual, a Delaunay triangulation. The Voronoi diagram divides the plane into cells (one per vertex), such that every point in the cell surrounding a vertex i has i as its nearest vertex. The structure is useful for nearest-neighbour location because the vertex $\mathcal{G}_i$ nearest to vertex to i is always in one of the (few, i.e. $\mathcal{O}(1)$) cells that share an edge with the cell of vertex i. An example is shown in Fig. 1. Voronoi diagrams for N points can be constructed with $\mathcal{O}(N \ln N)$ operations (see e.g. [24]), and the nearest neighbour identification for all N points can then be performed with a further $\mathcal{O}(N)$ operations.
- Dynamic insertion and removal of a point in the Voronoi diagram, and corresponding updating of all nearest neighbour information, can be performed with $\mathcal{O}(\ln N)$ operations [25] (to be repeated N times).
- The array of $d_{i\mathcal{G}_i}$ changes only by $\mathcal{O}(1)$ entries per iteration. Therefore one can represent it with a binary tree structure, whose construction requires $\mathcal{O}(N \ln N)$ operations and in which finding the minimal value, insertion and removal are all guaranteed to require at most $\mathcal{O}(\ln N)$. The binary tree is constructed once at start-up, and there are
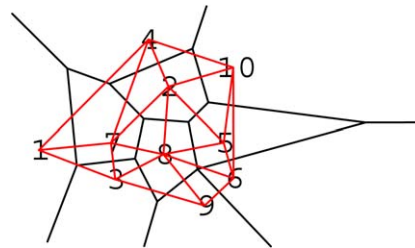

Fig. 1. The Voronoi diagram for ten random points. The Delaunay triangulation (red) connecting the ten points is also shown. In this example the points 1, 4, 2, 8 and 3 are the 'Voronoi' neighbours of 7, and 3 is its nearest neighbour.

then $\mathcal{O}(N)$ updates and searches for the minimum, leading to a total of $\mathcal{O}(N \ln N)$ operations.

Therefore both the geometrical and minimum-finding aspects of the k_t jet-finder can be related to known problems whose solutions require $\mathcal{O}(N \ln N)$ operations.

4. Timings

The program FastJet[5] codes concrete implementations of the N^2 and $N \ln N$ algorithms described above. It has been written in C++ and for the $N \ln N$ case makes use of a number of pre-existing components. Construction of a Voronoi diagram is a sufficiently common task (useful in areas of science ranging from astronomy to zoology) that several codes are publicly available. Of these, the only one that we found that also straightforwardly allows the addition and removal of points from a pre-constructed Voronoi diagram, was the Computational Geometry Algorithms Library (CGAL) [26], in particular its triangulation components [27].[6] For the binary tree we made use of a (red-black) balanced tree.[7]

Fig. 2 shows the running times for the two algorithms in FastJet as well as for KtJet, a standard implementation [22] of the N^3 algorithm. Our "N^2 algorithm" actually departs slightly from exact N^2 behaviour owing to certain further optimisations carried out.[8] The scaling with N of the Voronoi-based algorithm has been verified to go as $N \ln N$, as expected. It is the fastest algorithm only for $N \gtrsim 10^4$, owing to a large coefficient in front of its $N \ln N$ behaviour, mostly associated with the computational geometry tasks. This situation could conceivably be improved in the future by optimisations of the CGAL package or by replacing it with a dedicated implementation of the construction and updating of the Voronoi diagram.

---

[4] This last point is not strictly speaking trivial: when particle i is removed or merged we have to update the nearest neighbour information for all particles that previously had i as their nearest neighbour—fortunately one can show that on average, the number of particles that had i as a nearest neighbour is of $\mathcal{O}(1)$. One also needs to establish if any particles acquire the newly created particle ℓ as their nearest neighbour—this can be done in $\mathcal{O}(N)$ time by comparing each particle's current nearest neighbour distance with its distance to ℓ.

[5] Available from http://www.lpthe.jussieu.fr/~salam/fastjet.

[6] One issue relates to the fact that we need nearest-neighbour location on a cylinder (η–ϕ space) whereas CGAL works on the Euclidean plane. This problem can solved by making mirror copies of a small ($\sim 1/\sqrt{N}$) fraction of the vertices near the 0–2π border.

[7] Balanced trees are the basis of the map and set classes in the C++ standard template library.

[8] The coefficient of N^2 can be reduced by tiling the plane into rectangles of edge length $\geqslant 1$. Then for each vertex i one can limit the nearest neighbour search to its own tile and adjacent tiles—vertices further away will have $R_{ij} > 1$ and so $d_{ij} > d_{iB}$.
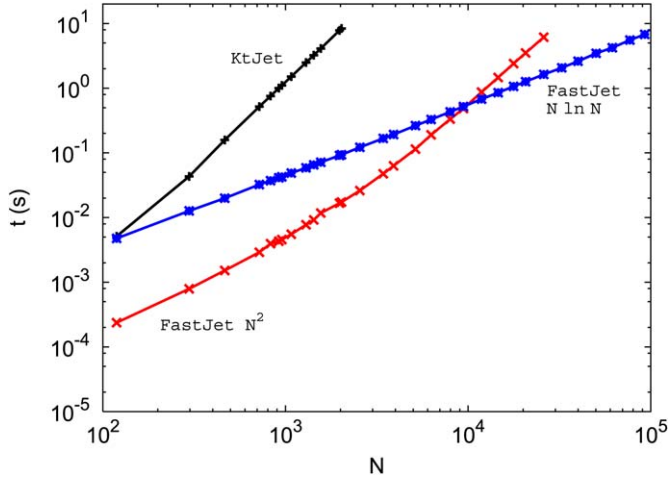
Fig. 2. The running times (on a 3 GHz Pentium 4 processor with 1GB of memory, 512KB of cache, and version 3.4 of the GNU g++ compiler) of the KtJet [22] and FastJet implementations of the k_t-clustering jet-finder versus the number of initial particles. Different values of N have been obtained by taking a LHC dijet event with $p_t \simeq 60$ GeV and adding on variable numbers of minimum bias events. Both kinds of events have been simulated with Pythia 6.3 [28].
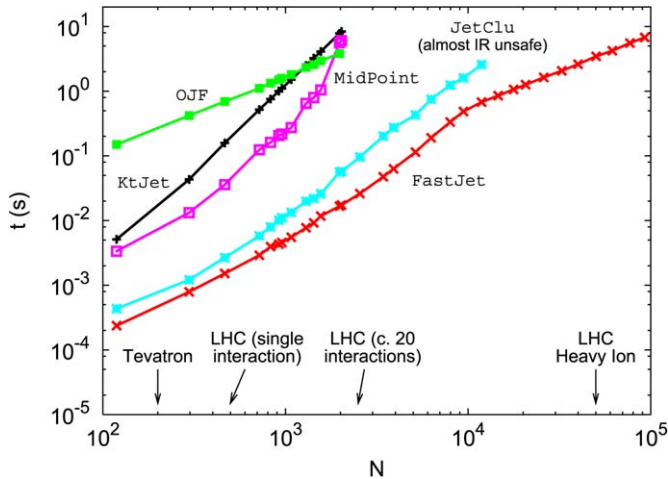

Fig. 3. The running times of various jet-finders versus the number of initial particles. JetClu is a widely-used cone-type jet-finder, however it is 'almost infrared unsafe', i.e. perturbative predictions have large logarithmic dependence on small parameters (e.g. seed threshold) [29,30]. MidPoint [29] is an infrared safe cone-type jet finder. For both we use code and parameters from CDF [31]. The optimal jet finder [9] (OJF) has been run with $\Omega_{cut} = 0.15$ and a maximum of 8 jets, so as to produce a final state similar to that returned by the k_t and cone jet-finders and to limit its run time.

The better of the N^2 and $N \ln N$ algorithms (which can be selected based on the value of N) runs at least an order of magnitude faster than the N^3 algorithm for all values of N shown, vastly more at large N.

Fig. 3 compares the running time of our combined N^2–$N \ln N$ FastJet implementation of the k_t jet-finder with other jet-finders whose code is publicly available. One sees that it runs at least an order of magnitude faster than all others (except the almost IR unsafe JetClu).

5. Perspectives

Since the FastJet algorithm is functionally equivalent to the standard N^3 algorithms used for the k_t jet finder till now, the results of the clustering are of course identical to those of other implementations. Howewer, its enhanced speed opens up new ways of using k_t clustering in the analysis of high-multiplicity events.

Historically, one apparent drawback of k_t-type jets with respect to cone-type jets in hadron–hadron collisions was considered to be the larger fluctuations of the areas of the jets defined by the clustering procedure. Such fluctuations would seem to make it more difficult to subtract, from the hard event, the energy coming from the non-perturbative underlying event and from any additional minimum bias interactions taking place in the same bunch crossing (pileup).

However, the fluctuations become irrelevant if one can easily estimate the area of *each individual* jet. This can be done on an event-by-event basis, as follows: because of the infrared safety of the k_t jet-finder algorithm, one can add a large number of extremely soft particles ("ghosts") to the event without modifying the properties of the hard jets. After clustering, each jet will contain a subset of the ghosts, and if the ghosts had a uniform density in η and ϕ, then the number of ghosts in a given jet will be a measure of its area. In practice we have found that the use of $\mathcal{O}(10^4)$ ghost particles is necessary to obtain reliable area estimations. This definition for the area of a k_t jet can of course be implemented with any coding of the jet-finder. It is however impractical, indeed nearly impossible, to deal with the required large number of ghost particles without a fast code.

Preliminary studies have shown that with simple assumptions about the uniformity of the underlying event and pileup, one can readily determine its size and subtract it from the hard jets, leading to good determinations of kinematical quantities (e.g. invariant masses) in high-luminosity pp collisions, or of single inclusive jet distributions in Pb–Pb collisions at the LHC. Full results will be shown in [32].

Two more observations are worth making before closing this section. They will both be discussed in more detail in [32].

The first is that it can also be interesting to examine alternative definitions of jet areas. One option is to make use of the areas of the Voronoi cells of all the real particles belonging to a given jet. This definition avoids the need to cluster thousands of ghost particles together with the real ones. It instead rests on the geometrical properties of the event, and on the computational geometry component of the FastJet implementation.

The second observation is that there exist clustering-type jet-finders other than the k_t jet-finder that share a large fraction of its features (including, of course, infrared safety), and the possibility of a fast implementation. One such example is the "Cambridge" jet-finder. It was originally formulated in the context of e^+e^- collisions in [33] and an inclusive version, adapted to hadron collisions, was given in [34]. We shall call this inclusive version the Cambridge/Aachen algorithm. It is defined in the same way as the k_t jet-finder at the beginning of Section 2, but replacing the particle–particle distance by $d_{ij} = R_{ij}^2/R^2$, and the particle–beam distance by $d_{iB} = 1$. We shall show in

[32] that the Cambridge/Aachen jet-finder has smaller average areas than the k_t jet-finder, making it perhaps even better suited for jet studies in high-multiplicity environments.

6. Conclusions

To conclude, we have identified an unexpected relation between clustering type jet-finders and a widely studied problem in computational geometry. The resulting reduction of the complexity of the k_t jet-finding problem, from N^3 to $N \ln N$, opens up the previously inconceivable option of using the k_t jet-finder for the large values of N that arise when considering all cells of a finely segmented calorimeter and for heavy-ion events. For moderate N, the one or two orders of magnitude in speed that we gain with a related N^2 approach pave the way to much wider use of the k_t jet finder for normal hadron-collider jet studies, especially at the LHC. More generally, the speed gains discussed in this Letter also suggest novel ways of using the k_t jet finder, which are the subject of ongoing investigation. One example, given in Section 5, is the determination of jet areas, knowledge of which is crucial for optimal subtraction of pileup contamination in high luminosity environments.

Acknowledgements

References

[1] G. Sterman, S. Weinberg, Phys. Rev. Lett. 39 (1977) 1436.
[2] See e.g., J. Womersley, Int. J. Mod. Phys. A 15S1 (2000) 607, eConf C990809 (2000) 607, hep-ex/9912009;
G.C. Blazey, B.L. Flaugher, Annu. Rev. Nucl. Part. Sci. 49 (1999) 633, hep-ex/9903058.
[3] See e.g., F. Abe, et al., CDF Collaboration, Phys. Rev. D 45 (1992) 1448;
G.C. Blazey, et al., hep-ex/0005012.
[4] W. Bartel, et al., JADE Collaboration, Z. Phys. C 33 (1986) 23;
S. Bethke, et al., JADE Collaboration, Phys. Lett. B 213 (1988) 235.
[5] S. Catani, Y.L. Dokshitzer, M. Olsson, G. Turnock, B.R. Webber, Phys. Lett. B 269 (1991) 432;
S. Catani, Y.L. Dokshitzer, M.H. Seymour, B.R. Webber, Nucl. Phys. B 406 (1993) 187;
S.D. Ellis, D.E. Soper, Phys. Rev. D 48 (1993) 3160, hep-ph/9305266.
[6] L. Lonnblad, Z. Phys. C 58 (1993) 471.
[7] Y.L. Dokshitzer, G.D. Leder, S. Moretti, B.R. Webber, JHEP 9708 (1997) 001, hep-ph/9707323;
M. Wobisch, Ph.D. thesis, RWTH Aachen, 1999, DESY-THESIS-2000-049.
[8] C.F. Berger, et al., eConf C010630 (2001) P512, hep-ph/0202207.
[9] D.Y. Grigoriev, E. Jankowski, F.V. Tkachov, Phys. Rev. Lett. 91 (2003) 061801, hep-ph/0301185;
D.Y. Grigoriev, E. Jankowski, F.V. Tkachov, Comput. Phys. Commun. 155 (2003) 42, hep-ph/0301226.
[10] L. Angelini, et al., Phys. Lett. B 545 (2002) 315, hep-ph/0207032;
L. Angelini, et al., Phys. Lett. B 601 (2004) 56, hep-ph/0407214.
[11] S. Chekanov, hep-ph/0512027.
[12] S. Moretti, L. Lonnblad, T. Sjostrand, JHEP 9808 (1998) 001, hep-ph/9804296.
[13] S.V. Chekanov, hep-ph/0211298.
[14] M.H. Seymour, Z. Phys. C 62 (1994) 127.
[15] M.H. Seymour, hep-ph/0007051.
[16] C. Buttar, et al., Les Houches physics at TeV colliders 2005, Standard Model, QCD, EW, and Higgs working group: Summary report, hep-ph/0604120 (Section 8).
[17] R.B. Appleby, M.H. Seymour, JHEP 0212 (2002) 063, hep-ph/0211426;
A. Banfi, M. Dasgupta, Phys. Lett. B 628 (2005) 49, hep-ph/0508159.
[18] J.M. Butterworth, B.E. Cox, J.R. Forshaw, Phys. Rev. D 65 (2002) 096014, hep-ph/0201098.
[19] V.M. Abazov, et al., D0 Collaboration, Phys. Lett. B 525 (2002) 211, hep-ex/0109041.
[20] V.M. Abazov, et al., D0 Collaboration, Phys. Rev. D 65 (2002) 052008, hep-ex/0108054.
[21] A. Abulencia, et al., CDF II Collaboration, Phys. Rev. Lett. 96 (2006) 122001, hep-ex/0512062.
[22] http://hepforge.cedar.ac.uk/ktjet/;
J.M. Butterworth, J.P. Couchman, B.E. Cox, B.M. Waugh, Comput. Phys. Commun. 153 (2003) 85, hep-ph/0210022.
[23] G.L. Dirichlet, J. Reine Ang. Math. 40 (1850) 209;
G. Voronoi, J. Reine Ang. Math. 133 (1908) 97;
See also: F. Aurenhammer, ACM Comput. Surveys 23 (1991) 345;
A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellations—Concepts and Applications of Voronoi Diagrams, second ed., John Wiley, 2000.
[24] S. Fortune, in: Proceedings of the Second Annual Symposium on Computational Geometry, 1986, p. 312.
[25] O. Devillers, S. Meiser, M. Teillaud, Comput. Geom.: Theory and Applications 2 (1992) 55;
O. Devillers, cs.CG/9907023.
[26] A. Fabri, et al., Softw. Pract. Exp. 30 (2000) 1167.
[27] J.-D. Boissonnat, et al., Comput. Geom. 22 (2001) 5.
[28] T. Sjostrand, et al., Comput. Phys. Commun. 135 (2001) 238, hep-ph/0010017;
T. Sjostrand, et al., hep-ph/0308153.
[29] S.D. Ellis, D.E. Soper, H.-C. Yang, 1993, unpublished.
[30] M.H. Seymour, Nucl. Phys. B 513 (1998) 269, hep-ph/9707338.
[31] http://www.pa.msu.edu/~huston/Les_Houches_2005/Les_Houches_SM.html.
[32] M. Cacciari, G.P. Salam, in preparation.
[33] Y.L. Dokshitzer, G.D. Leder, S. Moretti, B.R. Webber, JHEP 9708 (1997) 001, hep-ph/9707323.
[34] M. Wobisch, T. Wengler, hep-ph/9907280;
M. Wobisch, Measurement and QCD analysis of jet cross sections in deep-inelastic DESY-THESIS-2000-049.