CBM-SOFT-note-2005-002
I3HP-FutureDAQ-note-2005-003
22 September 2005

Elastic net for standalone RICH ring finding

S. Gorbunov[1] I. Kisel[2]

[1]DESY Zeuthen, Platanenallee 6, 15738 Zeuthen, Germany
[2]Kirchhoff Institute of Physics, Ruprecht-Karls University of Heidelberg, 69120 Heidelberg, Germany

Abstract

The elastic neural net is implemented for finding rings in RICH detector. The method does not require any prior track information and can be used for triggering. Application of the method to the RICH detector of the CBM experiment [1] shows very good efficiency and extremely high speed. The source code of the algorithm is given in the Appendix.

1 Introduction

The elastic net method [2] is a kind of artificial neural network [3, 4] that has been used for track recognition in high energy physics [5, 6, 7, 8, 9]. Based on the elastic net we have developed an algorithm for standalone RICH ring reconstruction.

2 The traveling salesman problem

The method is well illustrated on a simple example of the traveling salesman problem (TSP). The traveling salesman problem is a classic problem in the field of combinatorial optimization, in which efficient methods for maximizing or minimizing a function of many independent variables is sought. The problem is to find for a number of cities with given positions the shortest tour in which each city is visited once.

All exact methods known for determining an optimal route require a computing effort that increases exponentially with the number of cities, so in practice exact solutions can be attempted only on problems involving a few hundred cities or less. The traveling salesman problem thus belongs to the large class of nondeterministic polynomial time complete problems. Many heuristic algorithms were developed for the TSP aiming to bypass the combinatorial difficulties [10]. One of the most successful approaches to the problem is the elastic net of Durbin and Willshaw [2]. The elastic net can be thought of as a number of beads connected by elastics to form a ring. The essence of the method is to iteratively elongate a circular close path in a non uniform way until it eventually passes sufficiently near to all the cities to define a tour.

Following the deformable template approach [3], let us denote the cities by $\vec{x}_i$. We are going to match these cities with template coordinates $\vec{y}_a$ such that $\sum_a |\vec{y}_a - \vec{y}_{a+1}|$ is minimum and that each $\vec{x}_i$ is matched by at least one $\vec{y}_a$. Define a binary neuron s_{ia} to be 1 if a is matched

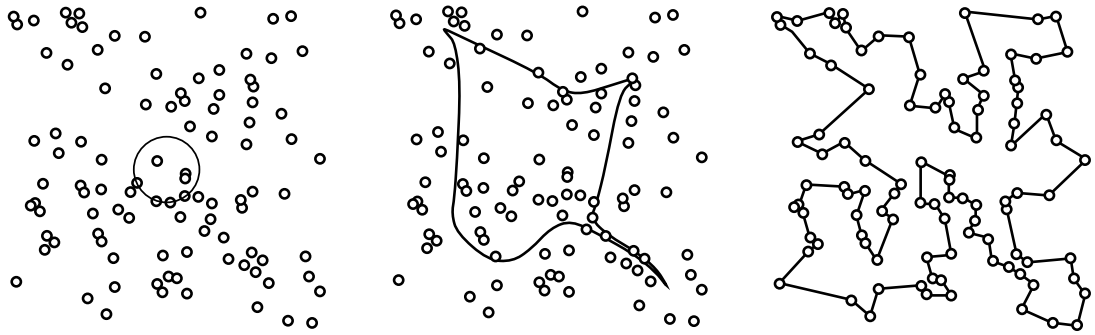

Figure 1: Example of the progress of the elastic net method in the traveling salesman problem with 100 cities [2]

to i and 0 otherwise. The following energy expression is to be minimized in a valid tour:

$$E\left(s_{ia}, \vec{y}_a\right) = \sum_{i,\,a} s_{ia} \cdot \left|\vec{x}_i - \vec{y}_a\right|^2 + \gamma \cdot \sum_{a} \left|\vec{y}_a - \vec{y}_{a+1}\right|^2 . \tag{1}$$

The multiplier γ governs the relative strength between matching and tour length. Applying the mean field approximation [3] one can derive the dynamical equation:

$$\Delta\vec{y}_a = \eta \left[2 \sum_{i} v_{ia} \cdot \left(\vec{x}_i - \vec{y}_a\right) + \gamma \cdot \left(\vec{y}_{a+1} - 2\vec{y}_a + \vec{y}_{a-1}\right) \right] , \tag{2}$$

where continuous neurons v_{ia} describe matching of a to i:

$$v_{ia} = \frac{e^{-|\vec{x}_i - \vec{y}_a|^2/T}}{\sum_b e^{-|\vec{x}_i - \vec{y}_b|^2/T}} . \tag{3}$$

Here the "temperature" T is decreasing at each update of templates $\vec{y}_a$, and η is the parameter controlling the minimization speed.

The algorithm is thus a procedure for the successive recalculation of the positions of a number of points of the plane in which the cities lie. The points describe a closed path which is initially a small circle centered on the middle of the distribution of cities and is gradually elongated non-uniformly to eventually pass near all the cities and thus define a tour around them, see Fig. 1 (for details see the original paper [2]). Each point on the path moves under the influence of two types of force (see Eq. 2):

1. the first moves it towards those cities to which it is nearest;
2. the second pulls it towards its neighbors on the path, acting to minimize the total path length.

By this process, each city becomes associated with a particular section on the path. The tightness of the association is determined by how the force contributed from a city depends on its distance, and the nature of this dependence changes as the algorithm progresses. Initially all cities have roughly equal influence on each point of the path. Subsequently a larger distance becomes less favored and each city gradually gets more influenced on the points on the path closest to it.

The elastic net algorithm produces tours of the same quality as other well known heuristic algorithms [11].

3 Discrete elastic net

The evolution of the elastic net coordinates in the continuous space results in a significantly large number of iterations without changing the order of the cities. This can be avoided if the net nodes are forced to coincide with cities at each iteration. Such a modification of the elastic net has been developed by us in order to increase the speed of the net. In this so-called discrete algorithm the elastic net can be represented as a closed tour passing exactly through a subset of the cities. An iteration consists of adding new cities to or releasing some cities from the net.

File name	Number of cities	Extra path (%)	Time, ms	Time per city, μs
berlin52	52	0.00	0.98	19
st70	70	4.27	1.27	18
kroA100	100	3.03	1.46	15
lin105	105	0.78	1.84	18
ch130	130	5.59	2.56	20
tsp225	225	5.34	4.36	19
pcb442	442	8.37	12.35	28
pr1002	1002	6.12	24.94	25
pr2392	2392	8.42	58.53	24

Table 1: Extra path length (in % to the optimum) and time of the discrete ENN algorithm in the TSP problem for several distributions of cities with known optimal tour

Results of application of the discrete ENN algorithm to several distributions of cities with known optimal tour are presented in table 1. The algorithm has good performance for real-life applications and is extremely fast. The last column of the table shows an almost constant execution time per city, which means linear behavior of the algorithm in spite of the increase of the combinatorial complexity of the problem.

The task of ring finding in the RICH detector with about 1000 hits per event is similar in combinatorial complexity to the `pr1002` example in the Table 1. Having now additional knowledge of the form of the final tour one can expect an increase of the speed down to a few milliseconds per event.

4 Elastic net method for ring finding

The problem of ring finding can be formulated as reconstruction of rings by measured hits registered in detection plane of RICH detector. The problem is complicated by overlapping of rings and presence of noise hits (see an example of a detected event in figure 2).

Here we describe a standalone ring finder which does not need neither the number of rings nor their parameters. The algorithm searches for rings consequently one by one in local areas, where noise hits and hits from other rings are also present. The main problem therefore is to construct single rings from maximum possible number of hits around the ring within the distance determined by the measurement errors.

The idea of the algorithm is to represent a ring as an elastic net evolving in the detector plane. Starting at some initial position the net iterates and converges to the group of ring hits.

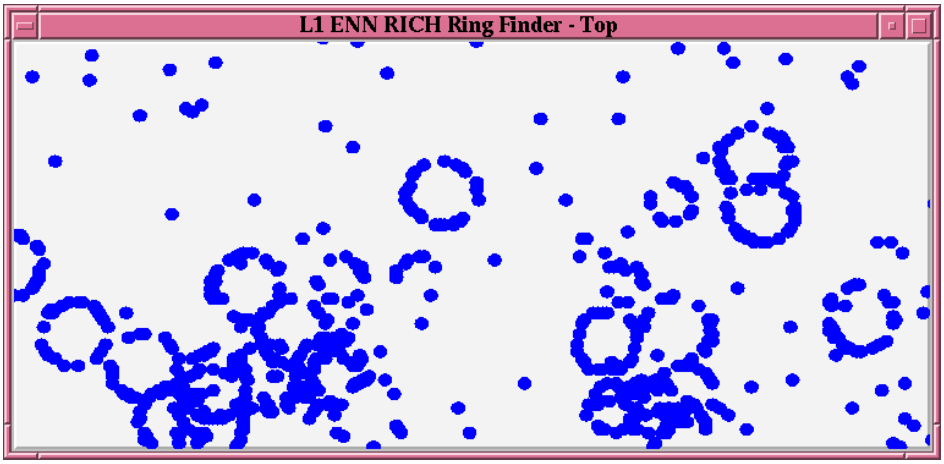

Figure 2: An event in the upper part of the RICH detector of the CBM experiment. The size of hits (registered photons) corresponds to the error of measurement.

Here, in contrast to the TSP problem, the circular form of rings is predefined. It is convenient to represent the net geometrically not as a set of nodes, but as a continuous curve in order to keep the circular form. Since there are no preferences on the ring parameters, the first evolution rule can be formulated as:

- the net has no internal forces.

In case of special preferences on the ring parameters (such as elliptical form) additional constraints can be easily introduced.

The second distinction from TSP is that the desired ring does not pass through all hits but only through a maximum possible number of them. Therefore the problem of single ring finding is divided into two steps: recognition of ring hits and further reconstruction of ring parameters.

At the recognition step a special energy function is used to separate ring hits from others.

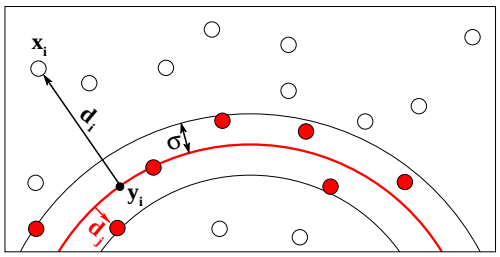

Figure 3: Selection of ring hits at the recognition step

Let us define the following variables (figure 3):
$\mathbf{x}_i$ — the i-th hit;
$\mathbf{y}_i$ — the elastic net point closest to the i-th hit;
$\mathbf{d}_i = (\mathbf{x}_i - \mathbf{y}_i)$ — distance between the hit and the net.

With these variables:

- the external forces are defined by minimizing the sum of (not squared) distances to hits

$$E = \sum |\mathbf{d}_i| \,, \tag{4}$$

therefore the net point $\mathbf{y}_i$ is attracted with the force

$$\Delta\mathbf{y}_i = -\mathbf{d}_i/|\mathbf{d}_i|\,. \tag{5}$$

Under the condition (5) the attraction force does not increase with increasing of the distance between the hit and the net.

The algorithm based on the energy function (4) is robust since hits at large distances practically do not bias the net. In other words, such force separates "signal" hits from "noise" hits.

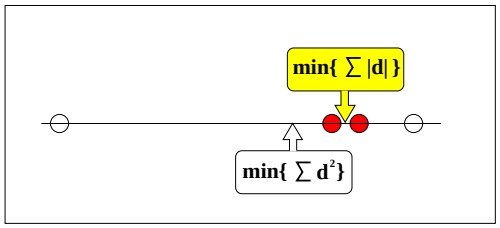

Figure 4: Energy function minima for the one-dimensional problem

Let us illustrate the robustness of the external force (5) on an one-dimensional example (Fig. 4), where hits are $\{x_i\}$ points and the net is just one point y. The task is to find a cluster with maximum number of hits. It can be seen that minimization of $E = \sum|x_i - y|$ provides the correct solution while the minimum of $E = \sum(x_i - y)^2$ is shifted to the left by a noise hit.

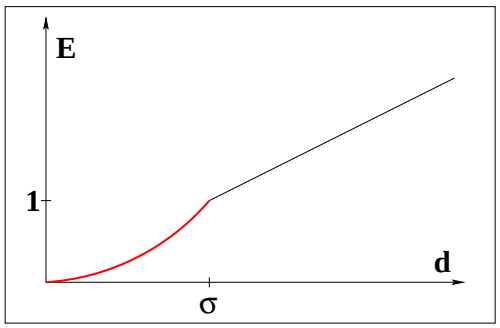

Figure 5: Dependence of the energy function on distance between the hit and the net

After the ring hits are recognized, one can apply the standard χ^2 minimization in order to get the optimal ring parameters.

The energy function E which combines both recognition and fitting steps can be represented as:

$$E = \sum \frac{|\mathbf{d}_i|^2}{\max\{\sigma, |\mathbf{d}_i|\}}\,, \tag{6}$$

where σ is the maximum deviation of hits from the ring.

The energy function (6) is shown in Fig. 5. The function has no gap at the border ($|\mathbf{d}_i| = \sigma$) and is equivalent to the χ^2 function for all hits satisfying the criteria ($|\mathbf{d}_i| \le \sigma$).

The ring is now not only recognized by the elastic net, but is also fitted by its hits making the final refit unnecessary. In addition, the use of the χ^2 function at small distances provides a reliable minimization procedure which converges after few iterations.

The main features of the ring finder can be summarized as following:

- the net is a continuous curve (a ring);

- there are no internal forces;
- the minimized energy function is

$$E = \sum \frac{|\mathbf{d}_i|^2}{\max\{\sigma, |\mathbf{d}_i|\}} \,. \tag{7}$$

An example (figure 6) of finding a circle illustrates the process of straightforward minimization of the energy function (7). The net starts with an arbitrary circle (top left), finds the mean circular distribution of all hits (top right) and finally converges to the solution (bottom left, compare with the simulated rings at bottom right).

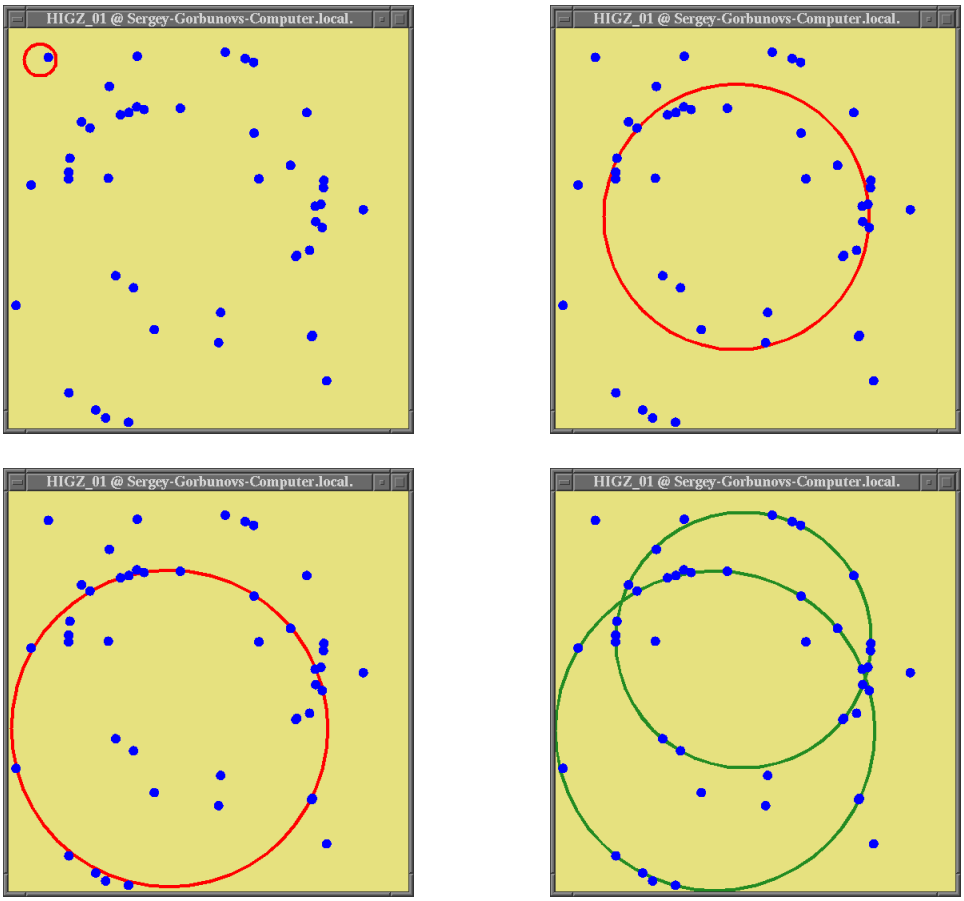

Figure 6: Finding of a circle with 15 hits by the elastic net in the presence of another overlapping ring with 15 hits and additional 15 noise hits randomly distributed

5 Application of the ring finder in the CBM experiment

A typical event registered in the RICH detector has many rings. In a standalone approach one assumes that number of rings and ring parameters are initially unknown.

Since rings in the RICH detector are independent on each other, one can also search for rings independently, one by one. To do this the elastic net method described in the previous section is applied to find single rings in local areas.

Most hits belong to rings, therefore ring search is performed in local areas around every hit assuming that the ring passes through the hit. The size of the search area is limited by the expected maximum ring size.

The ring finding procedure consists of a single loop over all hits:

- take a hit;
- initialize the search area around the hit;
- find a ring which passes through the hit;
- store the ring if found.

At the end a special ring selection procedure must be applied to remove clone rings and reduce the ghost rate.

Finding a single ring

In the RICH detector of the CBM experiment rings are well described by circles. Therefore the ring we are looking for is a circle which contains the current hit and has parameters (X, Y, R).

Let us shift the center of the coordinate system to the current hit. Since in the new coordinate system the circle has to pass through the $(0, 0)$ point, its radius is defined by the position of its center:

$$R = \sqrt{X^2 + Y^2}\,. \tag{8}$$

The ring has only two unknown parameters (X, Y) which have to be determined.

Following the general formula (Eq. 6) the energy function is:

$$E(X, Y) = \sum \frac{(\sqrt{(x_i - X)^2 + (y_i - Y)^2} - R)^2}{\max\{\sigma, |\sqrt{(x_i - X)^2 + (y_i - Y)^2} - R|\}}\,, \tag{9}$$

where (x_i, y_i) is position of the i-th hit.

The energy function (9) can be modified to accelerate the minimization procedure:

$$E(X, Y) = \sum \frac{((x_i - X)^2 + (y_i - Y)^2 - R^2)^2}{\max\{\sigma^2, |(x_i - X)^2 + (y_i - Y)^2 - R^2|\}}\,. \tag{10}$$

In addition, let us introduce weights w_i:

$$w_i(X, Y) = 1/\max\{\sigma^2, |(x_i - X)^2 + (y_i - Y)^2 - R^2|\}\,. \tag{11}$$

The final form of the energy function is:

$$E(X, Y) = \sum w_i(X, Y) \cdot ((x_i - X)^2 + (y_i - Y)^2 - R^2)^2\,. \tag{12}$$

The energy function E is minimized iteratively using weights w_i, calculated at the previous iteration:

Appendix 140-164

$$E(X^k, Y^k) = \sum w_i(X^{k-1}, Y^{k-1}) \cdot \left((x_i - X^k)^2 + (y_i - Y^k)^2 - (X^k)^2 - (Y^k)^2\right)^2\,. \tag{13}$$

The minimization of (Eq. 13) is similar to a weighted circle fit and takes 3–4 iterations to converge to the minimum.

Noise rejection

A noise rejection procedure is introduced to help the finder in complicated situations. At each iteration hits which are far away from the found ring ("noise" hits) are excluded from the search area.

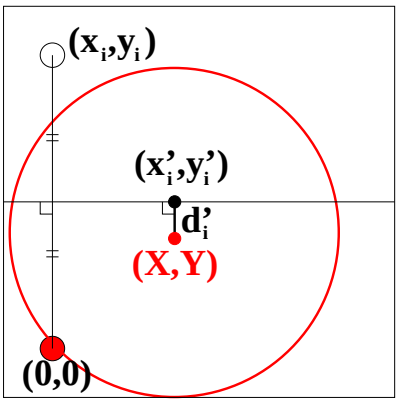

Figure 7: Noise rejection criteria d'_i for the hit (x_i, y_i)

It is useful to eliminate noise hits in the parameter space (XY). For every hit (x_i, y_i), position (x'_i, y'_i) of the preferred ring center (figure 7) is defined as:

$$(x'_i, y'_i) = (x_i, y_i) \cdot \left[\frac{1}{2} - \frac{x_i \cdot X + y_i \cdot Y}{x_i^2 + y_i^2}\right] + (X, Y)\,. \tag{14}$$

Distance in the parameter space between the hit and the ring is: 153

$$d'_i = ||(x'_i, y'_i) - (X, Y)|| = \sqrt{{x_i}^2 + {y_i}^2} \cdot \left|\frac{1}{2} - \frac{x_i \cdot X + y_i \cdot Y}{x_i^2 + y_i^2}\right|. \tag{15}$$

For hits which are close to the ring d'_i is set to zero. After each iteration a set of noise hits is determined using the criteria 154

$$d'_{noise} \geq C_{reject} \cdot \max_i\{d'_i\}\,, \tag{16}$$

and these hits are excluded from the searching area. The rejection criteria $C_{reject} = 0.5$ is used in the algorithm.

Fig. 8 shows the search area and the parameter space at different iterations. The ring hits in the parameter space are concentrated at the center even when the ring is not yet found.

Full scheme of ring finding

The ring finder is simple and consists of a single loop over hits: 049-238

1. take a hit h_k; 053
2. shift the center of the coordinate system to h_k; 078-080
3. initialize a search area around h_k; 072-120
4. find ring center (X, Y) by minimizing the energy function E (Eq. 13); 125-132

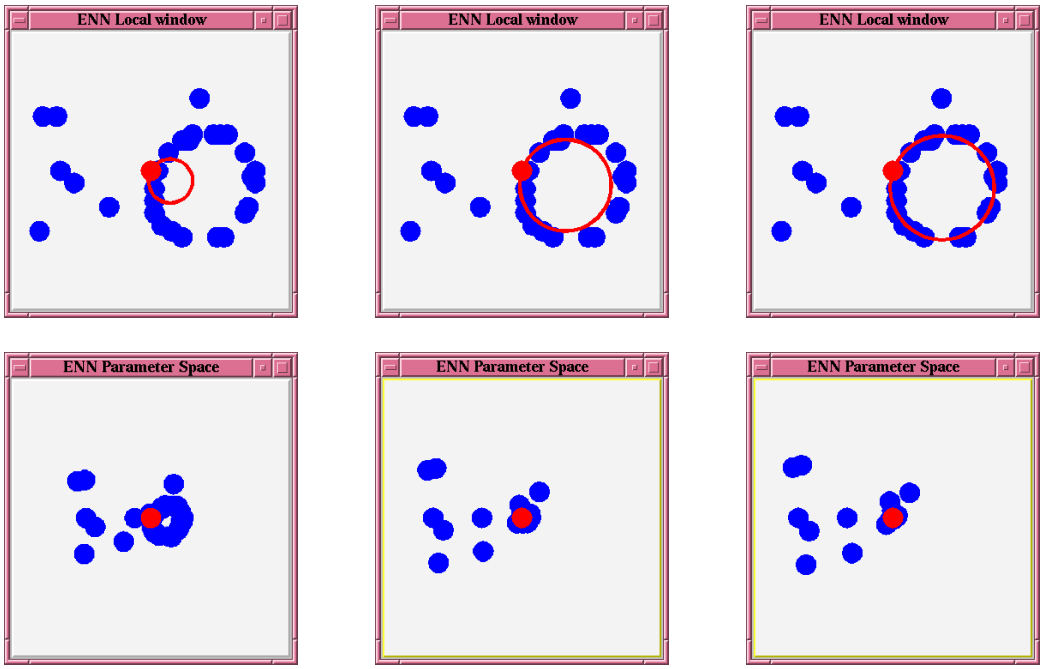

Figure 8: The ring search area (top) and the parameter space (bottom) during evolution (from left to right)

5. reject noise hits according to (Eq. 16) and continue to 4; 153,154
6. if no noise hits are left, store the found ring and mark its hits as used; 167-237
7. continue to 1.

The task of the described ring finder is to find the best ring-candidates in the local search areas.

Selection of found rings

After all ring-candidates are found in the local search areas a global optimization is performed by a ring selection procedure. The selection is based on a ring quality which includes: 240-286

- total number of hits on the ring N_{tot} ($\geq N_{tot}^{min}$);
- number of its own hits which are not on the already selected rings N_{own} ($\geq N_{own}^{min}$);
- purity of the ring $P = N_{own}/N_{tot}$ ($\geq P^{min}$).

Using the ring quality the algorithm takes the best ring, stores it and marks its hits as used. Selection repeats until the next best ring satisfies the quality criteria.

Timing

Since the net converges within 2–3 iterations, the total time to reconstruct an event is proportional to:

$$T_{\mathrm{ENN}} \sim N_{\mathrm{rings}} \cdot N_{\mathrm{hits\ per\ ring}} + N_{\mathrm{noise\ hits}}. \quad (17)$$

The selection time is negligible. The total time per event is about 5 ms for central Au+Au collisions at 35 AGeV.

6 Performance of the ENN ring finder

The ring finder has been applied in the CBM experiment. Aiming its implementation in a trigger we have focused on maximizing the speed of the algorithm.

Rings set	Performance (%)	Number of rings
Reference set efficiency	94.3	16
All set efficiency	74.0	54
Extra set efficiency	65.5	38
Clone rate	0.8	0
Ghost rate	12.8	7
Hits/event	1394	
Found MC rings/event	39	
Time/event	5.4 ms	
Time/hit	3.9 μs	

Table 2: Performance of the ENNRingFinder algorithm taken on central Au+Au collisions at 35 AGeV

The performance of the algorithm is presented in Table 2. The "all set" contains rings with 5 and more hits. In the "reference set" we put rings which originate from the target region and have 15 and more hits. The other rings form the "extra set".

A reconstructed ring is assigned to a generated Monte Carlo ring if there is at least 70% hits correspondence. A Monte Carlo ring is regarded as found if it has been assigned to at least one reconstructed ring. If the ring is found more than once, all additionally reconstructed rings are regarded as clones. A reconstructed ring is called ghost if it is not assigned to any Monte Carlo ring using 70% criteria.

The algorithm shows a high efficiency for reference rings (94%). The ghost rate can be further suppressed at the next step when rings will be matched to tracks from other detectors.

7 Conclusion

We have developed the standalone ring finder based on the elastic net method. Implemented for the CBM experiment, it has demonstrated a good ring finding efficiency and reliability. Additional track information can be included at the stage of ring finding in the parameter space improving the performance and the speed of the algorithm. Because of its computational simplicity and extremely high speed, the algorithm is considered to be further implemented in hardware which can increase the speed by another few orders of magnitude.

8 Acknowledgements

We acknowledge the support of the European Community-Research Infrastructure Activity under the FP6 "Structuring the European Research Area" programme (HadronPhysics,

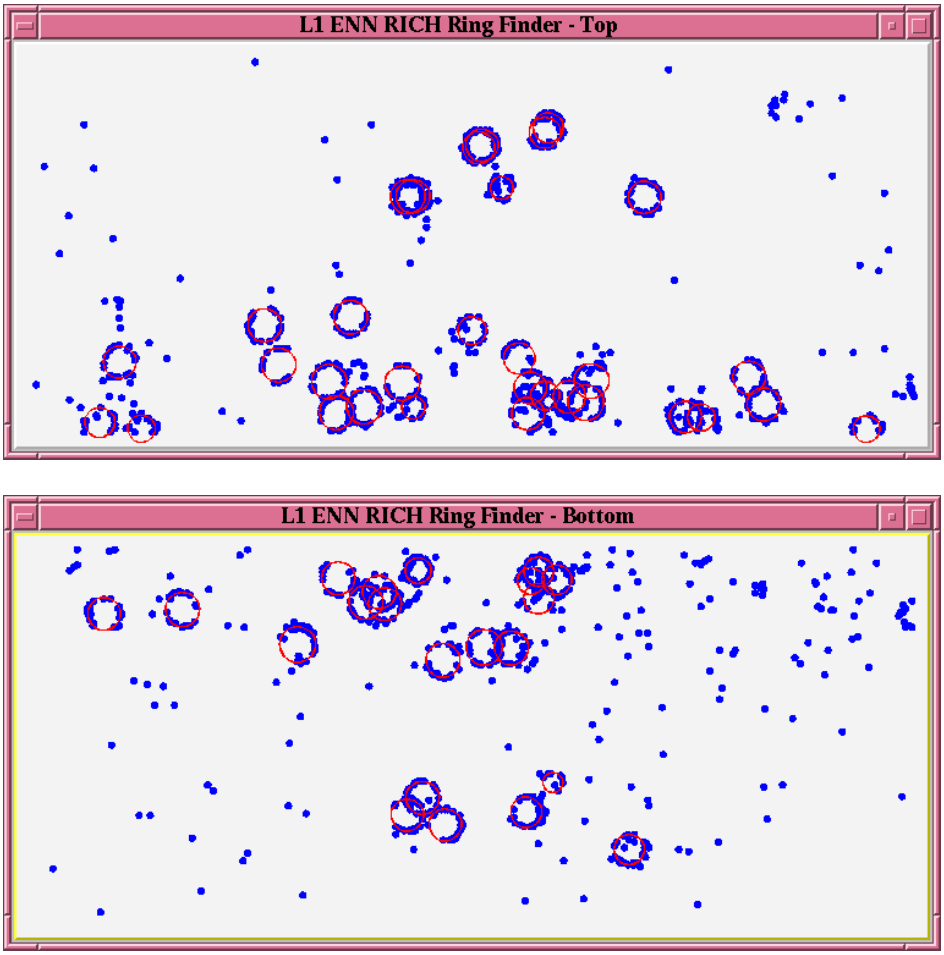

Figure 9: Example of a RICH event reconstructed by the ENNRingFinder algorithm. Top and bottom parts of the RICH detector are shown separately.

contract number RII3-CT-2004-506078).

References

[1] CBM Collaboration, "Compressed Baryonic Matter Experiment. Technical Status Report", GSI, Darmstadt, 2005,
(`http://www.gsi.de/onTEAM/dokumente/public/DOC-2005-Feb-447_e.html`).

[2] R. Durbin and D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method", *Nature* **326**, 16 April (1987) 689.

[3] C. Peterson and T. Rögnvaldsson, "Introduction to Artificial Neural Networks", *1991 CERN School of Computing*, Ystad, Sweden, 23 August – 2 September 1991, **CERN 92-02**, 1992, p. 113.

[4] I. Kisel, V. Neskoromnyi and G. Ososkov, "Applications of Neural Networks in Experimental Physics", *Phys. Part. Nucl.* **24** (6), November–December 1993, p. 657.

[5] I. Kisel, V. Kovalenko, F. Laplanche et al. (NEMO Collaboration), "Cellular Automaton and Elastic Net for Event Reconstruction in the NEMO-2 Experiment". *Nucl. Instr. and Meth.* **A387** (1997) 433.

[6] I. Abt, D. Emeliyanov, I. Gorbounov and I. Kisel, "Cellular automaton and Kalman filter based track search in the HERA-B pattern tracker", *Nucl. Instr. and Meth.* **A490** (2002) 546.

[7] M. Gyulassy and M. Harlander, "Elastic Tracking and Neural Network Algorithms for Complex Pattern Recognition", *Comp. Phys. Commun.* **66** (1991) 31.

[8] M. Ohlsson, C. Peterson and A.L. Yuille, "Track Finding with Deformable Templates — the Elastic Arms Approach", *Comp. Phys. Commun.* **71** (1992) 77.

[9] S. Gorbunov, I. Kisel and V. Tretyak, "Ring recognition method based on the elastic neural net", *Computing in High Energy Physics CHEP'01*, Beijing, China, 2001.

[10] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan and D.B. Shmoys, *The Traveling Salesman Problem*, John Wiley & Sons Ltd, 1985.

[11] I. Antoniou, S. Gorbunov, V. Ivanov, I. Kisel and E. Konotopskaya, "Elastic neural nets for the traveling salesman problem", *J. Comp. Meth. in Sci. and Eng.* **2** (2002) 111.

Appendix

```
001 #ifndef _ENNRingFinder_
002 #define _ENNRingFinder_
003
004 #include <vector.h>
005
006 struct ENNHit {
007   double x, y;               // coordinates
008   int busy;                  // quality of the best ring with this hit
009   // variables for local search:
010   double lx, ly, lr2;        // local coordinates
011   double S0, S1, S2, S3, S4; // coefficients for calculation of E
012   double C, Cx, Cy;          // coefficients for the parameter space
013   bool on_ring;              // is the hit close to the current ring
014 };
015
016 bool CompareENNHits( const ENNHit &h1, const ENNHit &h2 ){
017   return ( h1.x < h2.x );
018 }
019
020 struct ENNRing {
021   bool on;                   // is the ring selected?
022   double x, y, r;            // parameters
023   double chi2;               // chi^2
024   vector<ENNHit*> Hits;      // pointers to ring hits
025   // variables for the selection procedure:
026   int NHits;                 // number of ring hits
027   int NOwn;                  // number of its own hits
028   bool skip;                 // skip the ring during selection
029 };
030
031 inline void ENNRingFinder( vector<ENNHit> &Hits, vector<ENNRing> &Rings,
032                            double HitSigma = 1., int MinRingHits = 5,
033                            double RMin = 2., double RMax = 6. ){
034   // INITIALIZATION
035
036   const double Rejection = .5;
037   const double R2Min = RMin*RMin, R2Max = RMax*RMax;
038   const double HitSize = HitSigma/2.;
039   const double HitSize4  = 4 * HitSize;
040   const double AreaSize  = 2 * ( RMax + HitSigma );
041   const double AreaSize2 = AreaSize * AreaSize;
042
043   typedef vector<ENNHit>::iterator iH;
044   typedef vector<ENNHit*>::iterator iP;
045
046   Rings.clear();
047   sort( Hits.begin(), Hits.end(), CompareENNHits );
048
```

```
049   // MAIN LOOP OVER HITS
050
051   iH ileft = Hits.begin(), iright = ileft, i = ileft;
052
053   for( ; i != Hits.end(); ++i ){
054
055     if( i->busy >= 1 ) continue;  // already found hit
056
057     double left  = i->x - AreaSize;
058     double right = i->x + AreaSize;
059
060     while( ileft->x < left ) ++ileft;
061     while( iright != Hits.end() && iright->x < right ) ++iright;
062
063     vector<ENNHit*> SearchArea;
064     vector<ENNHit*> PickUpArea;
065
066     double X = 0, Y = 0, R = 0, R2 = 0;
067     int    NRingHits = 1;
068     double Dmax      = 0.;
069     double S0, S1, S2, S3, S4, S5, S6, S7;
070     int    NAreaHits = 0;
071
072     { // initialize hits in the search area
073
074       S0 = S1 = S2 = S3 = S4 = 0.;
075
076       for( iH j = ileft; j != iright; ++j ){
077         if( j == i ) continue;
078         j->ly  = j->y - i->y;
079         if( fabs(j->ly) > AreaSize ) continue;
080         j->lx  = j->x - i->x;
081         j->S0  = j->lx * j->lx;
082         j->S1  = j->ly * j->ly;
083         j->lr2 = j->S0 + j->S1;
084         if( j->lr2 > AreaSize2 ) continue;
085         NAreaHits++;
086         if( j->busy >= 13 ){
087           PickUpArea.push_back( &*j );
088           continue;
089         }
090         SearchArea.push_back( &*j );
091
092         double &lr2 = j->lr2;
093         double lr = sqrt(lr2);
094         if( lr > Dmax ) Dmax = lr;
095
096         j->S2 = j->lx * j->ly;
097         j->S3 = j->lx * lr2;
098         j->S4 = j->ly * lr2;
```

```
099         j->C = -lr/2;
100
101         if( lr > 1.E-4 ){
102           double w = 1./lr, w2 = w*w;
103           j->Cx = w*j->lx;
104           j->Cy = w*j->ly;
105           S0 += w2*j->S0;
106           S1 += w2*j->S1;
107           S2 += w2*j->S2;
108           S3 += w2*j->S3;
109           S4 += w2*j->S4;
110         }else {
111           j->Cx = j->Cy = 0;
112           S0 += j->S0;
113           S1 += j->S1;
114           S2 += j->S2;
115           S3 += j->S3;
116           S4 += j->S4;
117         }
118       }
119       if( NAreaHits+1 < MinRingHits ) continue;
120     }// end of initialization of the search area
121
122     // loop for minimization of E and noise rejection
123
124     do{
125       // calculate parameters of the current ring
126       double tmp = S0*S1-S2*S2;
127       if( fabs(tmp) < 1.E-10 ) break;
128       tmp = 0.5/tmp;
129       X = (S3*S1 - S4*S2)*tmp;
130       Y = (S4*S0 - S3*S2)*tmp;
131       R2 = X*X + Y*Y;
132       R = sqrt( R2 );
133
134       double Dcut      = Dmax * Rejection;            // cut for noise hits
135       double RingCut   = HitSize4 * (  HitSize + R ); // closeness
136       S0 = S1 = S2 = S3 = S4 = 0.0;
137       NRingHits = 1;
138       NAreaHits = 0;
139       Dmax = -1.;
140       for( iP j = SearchArea.begin(); j != SearchArea.end(); ++j ){
141         double dx = (*j)->lx - X;
142         double dy = (*j)->ly - Y;
143         double d2 = fabs( dx*dx + dy*dy - R2 );
144         (*j)->on_ring = ( d2 <= RingCut );
145         if( (*j)->on_ring ){
146           NRingHits++;
147           S0 += (*j)->S0;
148           S1 += (*j)->S1;
```

```
149           S2 += (*j)->S2;
150           S3 += (*j)->S3;
151           S4 += (*j)->S4;
152         }else {
153           double dp = fabs( (*j)->C + (*j)->Cx*X + (*j)->Cy*Y );
154           if( dp > Dcut ) continue;
155           if( dp > Dmax ) Dmax = dp;
156           NAreaHits++;
157           double w = 1./d2;
158           S0 += w*(*j)->S0;
159           S1 += w*(*j)->S1;
160           S2 += w*(*j)->S2;
161           S3 += w*(*j)->S3;
162           S4 += w*(*j)->S4;
163         }
164       }
165     }while( Dmax > 0 && NRingHits + NAreaHits >= MinRingHits );
166
167     // store the ring if it is found
168
169     if( NRingHits < MinRingHits || R2 > R2Max || R2 < R2Min ) continue;
170
171     { // final fit of 3 parameters (X,Y,R)
172       int n = 1;
173       S0 = S1 = S2 = S3 = S4 = S5 = S6 = S7 = 0.0;
174       for( iP j = SearchArea.begin(); j != SearchArea.end(); ++j ){
175         if( !(*j)->on_ring ) continue;
176         S0 += (*j)->S0;
177         S1 += (*j)->S1;
178         S2 += (*j)->S2;
179         S3 += (*j)->S3;
180         S4 += (*j)->S4;
181         S5 += (*j)->lx;
182         S6 += (*j)->ly;
183         S7 += (*j)->lr2;
184         n++;
185       }
186       double s0 = S6*S0-S2*S5;
187       double s1 = S0*S1-S2*S2;
188       double s2 = S0*S4-S2*S3;
189       if( fabs(s0) < 1.E-6 || fabs(s1) < 1.E-6 ) continue;
190       double tmp = s1*(S5*S5-n*S0)+s0*s0;
191       double A = ( ( S0*S7-S3*S5 )*s1-s2*s0 ) / tmp;
192       Y = (s2 + s0*A )/s1/2;
193       X = ( S3 + S5*A - S2*Y*2 )/S0/2;
194       R2 = X*X+Y*Y-A;
195       if( R2 < 0 ) continue;
196       R = sqrt( R2 );
197     }// end of the final fit
198
```

```
199     if( R2 > R2Max || R2 < R2Min ) continue;
200
201     ENNRing tmp;
202     Rings.push_back( tmp );
203     ENNRing &ring = Rings.back();
204     ring.x = i->x+X;
205     ring.y = i->y+Y;
206     ring.r = R;
207     ring.Hits.push_back(&*i);
208     ring.NHits = 1;
209     ring.chi2  = 0;
210     for( iP j = SearchArea.begin(); j != SearchArea.end(); ++j ){
211       double dx = (*j)->lx - X;
212       double dy = (*j)->ly - Y;
213       double d = fabs( sqrt(dx*dx+dy*dy) - R );
214       if( d <= HitSigma ){
215         ring.chi2 += d*d;
216         ring.Hits.push_back(*j);
217         ring.NHits++;
218       }
219     }
220     for( iP j = PickUpArea.begin(); j != PickUpArea.end(); ++j ){
221       double dx = (*j)->x - ring.x;
222       double dy = (*j)->y - ring.y;
223       double d = fabs( sqrt(dx*dx+dy*dy) - ring.r );
224       if( d <= HitSigma ){
225         ring.chi2 += d*d;
226         ring.Hits.push_back(*j);
227         ring.NHits++;
228       }
229     }
230     if( ring.NHits < MinRingHits ){
231       Rings.pop_back();
232       continue;
233     }
234     ring.chi2 = ring.chi2 / ( ring.NHits - 3)/.3/.3;
235     for( iP j = ring.Hits.begin(); j != ring.Hits.end(); ++j ){
236       if( (*j)->busy<ring.NHits ) (*j)->busy = ring.NHits;
237     }
238   }// END OF THE MAIN LOOP OVER HITS
239
240   // SELECTION OF RINGS
241
242   typedef vector<ENNRing>::iterator iR;
243
244   for( iH i = Hits.begin(); i != Hits.end(); ++i ) i->busy = 0;
245   for( iR i = Rings.begin(); i != Rings.end(); ++i ){
246     i->skip = i->on = 0;
247     i->NOwn = i->NHits;
248     if( ( i->NHits < MinRingHits ) ||
```

```
249         ( i->NHits <= 6 && i->chi2  > .3 ) )
250       i->skip = 1;
251   }
252
253   do{
254     iR best = Rings.end();
255     int    bestOwn  = 0;
256     double bestChi2 = 1.E20;
257     for( iR i = Rings.begin(); i != Rings.end(); ++i ){
258       if( i->skip ) continue;
259       if( ( i->NOwn < 1.0*MinRingHits ) ||
260           ( i->NHits < 10 && i->NOwn < 1.00*i->NHits ) ){
261         i->skip = 1;
262         continue;
263       }
264       if( ( i->NOwn >  1.2*bestOwn ) ||
265           ( i->NOwn >= 0.8*bestOwn   &&  i->chi2 < bestChi2 ) ){
266         bestOwn  = i->NOwn;
267         bestChi2 = i->chi2;
268         best = i;
269       }
270     }
271     if( best == Rings.end() ) break;
272     best->skip = 1;
273     best->on   = 1;
274     for( iP i = best->Hits.begin(); i != best->Hits.end(); ++i )
275       (*i)->busy = 1;
276     for( iR i = Rings.begin(); i != Rings.end(); ++i ){
277       if( i->skip ) continue;
278       double dist = i->r+best->r+2*HitSigma;
279       if( fabs(i->x-best->x) > dist ||
280           fabs(i->y-best->y) > dist ) continue;
281       i->NOwn = 0;
282       for( iP j = i->Hits.begin(); j != i->Hits.end(); ++j ){
283         if( !(*j)->busy ) i->NOwn++;
284       }
285     }
286   }while(1);
287 }
288
289 #endif
```