ALEPH 90-148
SOFTWR 90-14
G. Stimpfl
18.10.1990

UAB-LFAE 90-06
August 90

Fast Track Finding with Neural Nets

Georg Stimpfl-Abele[(a)*], **Lluís Garrido**[(b)*]
(a) Department of Physics
Florida State University
Tallahassee FL 32306, USA
(b) Laboratori de Física d'Altes Energies
Universitat Autònoma de Barcelona
E-08193 Bellaterra (Barcelona), Spain

August 1990

Abstract

We have used a neural network (NN) technique for track reconstruction in a realistic environment. An algorithm based on an Hopfield-style recurrent NN was developed and tested on the track coordinates measured by the TPC of the ALEPH detector at LEP. The efficiency and time consumption are given and are compared with a conventional pattern recognition method. The performance of the algorithm for large numbers of tracks (up to 200), as expected for LHC and SSC detectors, is discussed.

(Submitted to Computer Physics Communications)
Accepted by

1 Introduction

The field of neural nets has developed very fast during the last years. People started to realize the variety of its applications like finding optimal solutions with constraints, pattern recognition and learning, with the remarkable possibility to use dedicated chips. Basic concepts of neural nets can be found in reference [1], and recent overview articles in reference [2].

A neural network consists of processing units, which are called neurons, and connections between them. The state (activation) of a neuron is a function of the

* Now at CERN

input which it receives from other neurons. The output of a neuron is usually equal to its activation. The state of each neuron depends therefore on the state of the neurons from which it receives input and the strength (weight) of these connections.

Neural nets can learn representations by changing the weights of the connections between the neurons [3]. Such learning algorithms are in general not well suited for track finding since we have an enormous number of different coordinate patterns. This number can be reduced dramatically by taking symmetries into account. But this is still a very difficult task. Instead mostly single-layer networks with constraints and constant weights are used [4] [5] [6].

2 Neural nets with constraints

2.1 General considerations

Single-layer networks are similar to spin-glasses [7] where each unit can be in one of two possible spin states $S_i = \pm 1$ (spin up or down) and where each spin interacts with the other spins. We can characterize the state of neurons in the same way, where $S = 1$ means activation and $S = -1$ deactivation. The connection strength (weight) between two neurons, denoted by T_{ij} for the pair of neurons i and j, can take both positive and negative values, and is in most of the models symmetric ($T_{ij} = T_{ji}$) with vanishing diagonal elements ($T_{ii} = 0$) .

The dynamics of the system is given by

$$S_i = sign(\sum_j T_{ij} S_j). \tag{1}$$

It is easy to show [8] that this local update rule evolves the system into a local minimum of the energy function

$$E(S) = -\frac{1}{2} \sum_{ij} T_{ij} S_i S_j \tag{2}$$

if Tij is symmetric.

In most of the applications we are not interested in local minima but in the global minimum. One way to reach the global minimum is the introduction of thermal noise to allow the system to leave a local minimum. In this case the state vector S is Boltzmann distributed with the probability

$$P(S) = \frac{1}{Z} e^{-E(S)/T}. \tag{3}$$

The partition function

$$Z = \sum_S e^{-E(S)/T} \tag{4}$$

is the sum over all possible states and T is the temperature.

We can calculate Z with the mean field theory (MFT) approximation [9] and get for the thermal average of S_i

$$V_i \equiv < S_i >_T = tanh\Big(-\frac{1}{T}\frac{\partial E}{\partial V_i}\Big) \tag{5}$$

where the energy is defined analogous to equation (2) as

$$E = -\frac{1}{2}\sum_{ij} T_{ij} V_i V_j \quad . \tag{6}$$

The global minimum can be found by iterative calculation of the activations V_i. We will use this approximation in the following.

V_i can take any value between -1 and 1. For many applications it is more convenient to limit the range to the interval [0,1] in order to get no contribution from deactivated neurons. In this case we make the linear transformation

$$V_i = \frac{1}{2}\Big[1 + tanh\Big(-\frac{1}{T}\frac{\partial E}{\partial V_i}\Big)\Big]. \tag{7}$$

This expression is equivalent to the 'logistic function'

$$V_i = \frac{1}{1 + e^{-X}} \tag{8}$$

with

$$X = -\frac{2}{T}\frac{\partial E}{\partial V_i} \, . \tag{9}$$

We can generalize the energy function by imposing constraints in the form

$$E = -\frac{1}{2}\Big(\sum_{ij}(T_{ij} - \alpha C_{ij})V_i V_j - \beta D\Big). \tag{10}$$

C is a constraint matrix, D a global constraint for all activations, and α and β are Lagrangian multipliers. Positive weights enforce the activation whereas the two constraint terms weaken it (therefore they are also called penalty terms).

2.2 Track finding

In track reconstruction we want to associate coordinates to tracks. We can regard a track with n coordinates as a set of $n - 1$ consecutive lines ('track segments') with a smooth shape and without bifurcation. These track segments are chosen as the neurons. The weights depend on the geometry (length of the lines, crossing angle between the lines, etc.). For the calculation of the energy terms we have to use the coordinates. Therefore we characterize in the following the neuron i (j) by the coordinate k (m) at which it starts and by the coordinate l (n) where it ends. Equation (10) becomes now

$$E = -\frac{1}{2}\Big(\sum_{klmn}(T_{klmn} - \alpha C_{klmn})V_{kl} V_{mn} - \beta D\Big). \tag{11}$$

Since we are only interested to connect neurons with one coordinate in common we have

$$T_{klmn} = \delta_{lm} T_{klln} \tag{12}$$

and we describe these matrix elements from now on by the indices of its 3 coordinates, T_{kln}.

The constraints are formulated in such a way that we do not associate a coordinate to more than one track (α term) and that we activate the expected number of neurons (β term). For convenience we limit the range of V_{kl} to the interval [0,1]. The inhibition terms can now be written in the form [6]

$$C_{klmn} = \delta_{km}(1 - \delta_{ln}) + \delta_{ln}(1 - \delta_{km}) \tag{13}$$

$$D = (\sum_{mn} V_{mn} - N_a)^2. \tag{14}$$

N_a is the expected number of activated neurons. Combining the last 4 equations yields

$$E = -\frac{1}{2}\Big[\sum_{kln} T_{kln} V_{kl} V_{ln} - \alpha(\sum_{kln(n\neq l)} V_{kl} V_{kn} + \sum_{klm(m\neq k)} V_{kl} V_{ml}) - \beta(\sum_{mn} V_{mn} - N_a)^2\Big]. \tag{15}$$

This means that we have in the α term for each neuron kl to sum over all the other neurons which start at the coordinate k or end at the coordinate l. Hence there is a sort of competition between the neurons starting or ending at the same coordinate and each neuron tries to switch off the competing neurons via this constraint term.

Combining equation (7) and (15) we finally get the following update rule for the neurons

$$V_{kl} = \frac{1}{2}\Big[1 + tanh\Big(\frac{1}{T}\sum_{n} T_{kln} V_{ln} - \frac{\alpha}{T}(\sum_{n\neq l} V_{kn} + \sum_{m\neq k} V_{ml}) - \frac{\beta}{T}(\sum_{mn} V_{mn} - N_a)\Big)\Big]. \tag{16}$$

2.3 Choice of the parameters

In order to get the system converge to a 'good' configuration we have to tune the parameters very carefully. We want that the activated neurons form chains with smooth shape. We have two sorts of parameters to achieve that: the parameters in the weight term and the lagrangian multipliers α and β in the constraints.

The weights have to be defined in such a way that we favour short and almost straight connections. Denby [4] and Peterson [6] suggested a connection matrix of the form

$$T_{kln} = \frac{\cos^{\lambda} \theta_{kln}}{d_{kl} + d_{ln}} \quad . \tag{17}$$

θ is the angle between the lines kl and ln and d the length of the lines (distance between the start and the end points). The parameter λ is given a rather large value (values up to 100 can be found in literature [5]).

α has to be set with care, since a too large value means that an activated neuron switches off all competing neurons. This usually leads to bad results since very often

wrong neurons are activated first. The biggest inconvenience of this effect is that we get very often a zigzag pattern instead of two rather close parallel tracks.

The β term should ensure that we activate the expected number of neurons. A disadvantage in pattern recognition is that we do not know this number exactly since there are always some coordinates which we can not associate to tracks and we need to know the number of tracks in advance since a track with n coordinates is made up by $n-1$ lines. But this term serves usually more as stimulation than as a constraint, since the the number of activated neurons is already limited by the α term. The stimulation comes from the fact that the β contribution becomes positive if $N_a > \sum_{mn} V_{mn}$. Hopfield for example treats, in his solution of the travelling salesman problem [8], N_a as a parameter which is fixed at a value which is 50 % higher than the correct one.

The temperature T determines the speed of the convergence (i.e. how fast the activations reach their asymptotic values 0 and 1). High temperature means slower convergence but gives the system more chances to escape from local minima than in the case of low temperatures.

2.4 Update of the neuron states

The dynamics of the network is determined by equation (16). We preset the activation values of the neurons randomly and we update them according to this formula until the values converge which corresponds to a minimum in the energy of the system. The symmetry of the weight matrix ensures that we reach such a minimum.

The update of the V_{kl} can in principle be done either synchronously (each neuron state is changed at time step t depending on the states of the other neurons at time step $t-1$) or asynchronously (we update one neuron state after the other, each neuron 'sees' therefore the other neurons in their actual state). In practice only the asynchronous update works, because in the synchronous case sometimes a situation arrives where the system finds two very different solutions and flips in each iteration from one solution to the other.

This is a consequence of the constraints and can easily be understood in the following way: if we update the states of the neurons synchronously then each neuron reacts according to the previous state of the other neurons. Starting the iterations with the usual rather small initial activation we get almost no contribution from the constraint terms and many connections are established via the weight matrix. In the next step each neuron gets a big penalty from the constraint terms because many neurons connecting to the same coordinates are on. Therefore almost all neurons are switched off. In the next iteration they are switched on again and so forth.

3 Fast track finding algorithm for realistic applications

We have developed a new method for track reconstruction based on the ideas outlined in the previous chapter. Our goal was to find a neural network algorithm which can compete with conventional methods concerning efficiency, memory usage and execution time on normal computers under realistic conditions. As such a test case we have choosen the coordinates measured by the TPC of the Aleph detector at LEP.

3.1 The Aleph TPC

We want to give here only a very short overlook over the detector for which our algorithm was tuned. The whole Aleph detector is described in much detail in [10]. The Aleph Time Projection Chamber (TPC) is a large three-dimensional imaging drift chamber. It is a cylinder with axial parallel magnetic and electric fields. The axis of the cylinder coincides with the beam axis which defines the Z direction of the Aleph coordinate system. The electric drift field points from each end-plate towards the central membrane that divides the chamber into two halves. The electrons produced by the ionization of traversing charged particles drift towards one end-plate, where they induce ionization avalanches in a plane of wire chambers. These avalanches are detected and give the impact point and the arrival time of the drifted electrons. This means that the TPC provides three-dimensional coordinates. The X and Y coordinate are given by the impact point and the Z coordinate is obtained from the measured drift time.

The trajectory of a charged particle inside the TPC is a helix, and its projection onto the end-plate is an arc of a circle. The magnetic field has a strength of 1.5 T.

The end-plates are divided into 21 pad-rows, the radius of the first pad-row is about 40 cm and the radius of the last pad-row about 180 cm. The radial distance between two pad-rows is roughly 6 cm. Each half of the TPC has a lateral distance (Z) of 220 cm. The coordinates are measured with a resolution of about 200 μm in XY and of about 2 mm in Z.

3.2 Definition of the neurons

In the Aleph TPC we have to cope with events with more than 500 coordinates. If we connect all these coordinates we get more than 2.5×10^5 neurons and more than 6×10^{10} connections between these neurons. To keep the size of the problem reasonable we apply several cuts.

First we make a quality cut on the coordinates and keep only coordinates which are reasonably well measured :

- $\sigma_{xy} < 0.3cm$
- $\sigma_z < 1.0cm$.

In this way we eliminate about 5% of the coordinates. The remaining coordinates are only connected by lines if

- the difference in phi is < 0.15 rad
- the difference in theta is < 0.10 rad
- the difference in the pad-row number is < 4 .

By applying these cuts we reduce the number of neurons by more than a factor 100. We connect the coordinate pairs only in one direction and we chose the coordinate with the bigger radius in the X-Y plane ($\rho = \sqrt{x^2 + y^2}$) as the starting point. The lines point therefore inwards.

3.3 Definition of the weight matrix

Since the coordinates in the Aleph TPC are real 3-dimensional points we modify equation (17) in the form

$$T_{kln} = \frac{\cos^{\lambda} \psi_{kln}}{d_{kl}^{\mu} + d_{ln}^{\mu}} \; . \tag{18}$$

ψ is the angle in space between the two lines and d_{kl} the distance between the coordinates k and l in XY. Since we do not want to give smaller weights to forward tracks we do not take the Z coordinate into account in the definition of d. The distances are normalized to the distance between two pad-rows to have maximal weights of the order of 1. We give the above weights to connections which fulfil the condition

$$cos\psi_{kln} > 0.90 \; .$$

All other weights are set to zero. We prefer not to have negative weights because it happens rather frequently that a neuron belonging to a track has several bad connections and it might get switched off if their contributions are negative. We rather want to eliminate such connections by the inhibition term α. Although we connect two points only in one direction (from bigger ρ to smaller ρ) we have to sum in the weight term for a given neuron kl over the contribution of all 'incoming' lines (ending at k) and all 'outgoing' lines (starting at l).

We found an efficient way to save execution time and memory, and to avoid zigzag patterns: after we have calculated the weights of all incoming and outgoing lines we only keep the best (i.e. connection with the biggest weight) incoming and the best outgoing line and set all the other weights to 0. This means that we have for each neuron only 2 weights to store.

This approach of a simple local pattern recognition works astonishingly well. We found it far superior over other methods which we tried out, for example a penalty term for line crossings. The time and memory demand of such a term is rather high and one has an additional parameter to tune.

3.4 Update of the neurons

We use the same α term as described in chapter 2 but instead of the rather weak and ambigous global contraint term β we prefer to have an input bias term B which has the same value for all neurons. We give B a small positive value to stimulate the activation of the neurons at the beginning, and we can lower its value towards the end of the iterations in order to switch off neurons which are no longer enforced by their neighbours. We also multiply the weight terms by the parameter c. This allows us to tune the interaction between the weights, the constraints and the bias with the parameters c, α and B and to treat the temperature as pure normalization factor to control the speed of the convergence. The update rule for the neurons (16) reads now

$$V_{kl} = \frac{1}{2}\Big[1 + tanh\Big(\frac{c}{T}\sum_{n} T_{kln} - \frac{\alpha}{T}(\sum_{n\neq l} V_{kn} + \sum_{m\neq k} V_{ml}) + \frac{1}{T}B\Big)\Big] \tag{19}$$

with the weights T_{kln} given in equation (18).

We update the neurons asynchronously starting at the outer boundery and we iterate this process until the system converges. We can measure the convergence either by the change in energy between two iterations or by the sum over the changes of the activations of all neurons. We used the criterium

$$\frac{1}{N}\sum_{kl} |V_{kl}^{I} - V_{kl}^{I-1}| \leq .0005 \tag{20}$$

to stop the iterations. N denotes the number of neurons and I the iteration number. Since we use a limited number of weights the system converges in general rather fast (in less than 10 iterations).

3.5 Memory usage

We want to give here an estimate for the memory size needed for our algorithm. Since we neither connect all points nor all lines we use vectors instead of full matrices mostly filled with 0. We use two-dimensional tables for the relation between entities. We need for example to know which lines enter or leave a given coordinate. We have a list of coordinates and two lists of lines, the first one ordered by the start point and the second one ordered by the end point. We relate coordinates and lines by a table which contains for each coordinate the number of lines and the pointer of the first line into the line list.

We have as usual to balance between memory usage and execution time. Since our program is very small we prefer to store some geometry constants in arrays instead of recalculating them several times.

We use 7 geometrical constants per coordinate (X, Y, Z, R, θ, ϕ, pad-row number). In addition we have a two-dimensional table for the incoming lines and another one for the outgoing lines. Hence we have to store at least 11 items per coordinate.

We have 2 lists of lines (ordered by the start and by the end coordinate, respectively). We store for each line the start and the end coordinate, the index of the

incoming and of the outgoing line which contributes to the weight term and their weights, and the activation. This makes 9 numbers per line.

If we allow for some temporary space we can estimate the total length of all arrays needed for an event with N_C coordinates and N_L lines by

$$L = 12 \times (N_C + N_L).$$

For an event of about 500 coordinates we have on average 2000 lines and we get $L = 3 \times 10^4$. This is an enormous reduction compared to the size of the full matrices, since the weight matrix alone would have $N_L^2 = 4 \times 10^6$ elements.

4 Reconstruction of real events

We tested the algorithm described in the last chapter on real events measured by the Aleph TPC at LEP and compared its performance with the standard Aleph event reconstruction program JULIA. For this purpose we have selected decays of the Z^0 into hadrons with more than 20 charged particles.

4.1 Performance of the algorithm

We used for our study the parameter set

$$c = 10, \quad \alpha = 5, \quad B = 0.2, \quad T = 1, \quad \lambda = 5, \quad \mu = 2$$

and achieved good quality of the results and fast convergence. The behaviour of the network does not depend strongly on the parameters and it is therefore easy to find appropriate values for them.

First we want to demonstrate the convergence process on a real event. Figures 1 and 2 show all neurons in the XY and the RZ projection for a real Z^0 event measured in the Aleph TPC. We see that well separated tracks are already almost 'reconstructed', except from overlapping lines. But we have many lines in regions with nearby tracks.

We let the system evolve and it converges after 6 iterations. Figures 3 and 4 show all neurons with an activation $> .9$. All tracks have been found properly.

We have studied the track finding efficiency of this NN algorithm. The efficiency for a track which has more than 6 reconstructed coordinates and which leaves the TPC is bigger than 99%.

The execution time for the setup of the neural net (definition of the neurons and calculation of the weight matrix) is comparable with the time spent in the iterations. From this we conclude that it makes no sense to use dedicated hardware for the iteration process since one is limited by the setup time.

4.2 Comparison with conventional methods

The Aleph reconstruction program uses for the track finding in the TPC a highly optimized program based on a three-dimensional track following strategy [11]. We use this code as reference.

The track finding efficiencies of both methods are comparable: about 99% for the NN method against 99.7% for the conventional method. The biggest difference is the number of wrongly associated coordinates. While this is a rather rare case for the conventional method this happens more frequently in the NN approach. It is clear that a neural network program which uses only soft constraints and very simple geometrical constants is not as good as a sophisticated algorithm based on hard constraints (fits). But the big advantage of neural networks is their flexibility, the small number of parameters and their capability to find solutions near the optimum.

A very important point is the amount of computing time needed. There are some estimates that for track finding neural networks need about 10 times more time than other methods. Our algorithm needs for a big event (500 coordinates) on average about 1.7 sec (normalized to a 1 MIPS computer) for the setup (definition of the lines and calculation of the weight and constraint 'matrices') and about 0.3 sec for each iteration. Since we reach convergence already after a few iterations the total execution time amounts to roughly 3.5 sec. This has to be compared with 3.6 sec needed by the conventional code. This means that our algorithm is competitive with the standard one. The timing is summarized in table 1.

	Average time per event (sec.)
NN (initialization)	1.7
NN (one iteration)	0.3
NN (total)	3.5
Conventional	3.6

TABLE 1
Time comparison

4.3 Scaling properties

For LHC and SSC detectors we expect about 200 tracks per event. We studied the dependence of our NN approach as a function of the number of tracks per event with 21 points per track. Figure 5 shows the time consumption as a function of the number of tracks for the NN and the conventional algorithm. The NN method can again compete with the standard one. Furthermore we see that more than 60% of the NN time is used for the initialization. Because we did not include noise in this study the total time for both methods is smaller than in the last section.

As an example of a big reconstructed event we show in figure 6 the generated lines for a Monte Carlo event with 100 tracks, and in figure 7 the activated lines after convergence.

4.4 Further improvements

We defined the weight matrix in a very simple way. One could do this in much more detail with a local pattern recognition method. But we think that this would lead to a big overhead in execution time and one would not gain much, since neural networks tend to find a good solution rather than the best solution.

The time spent in the setup of the iterations is even in our very simplified program quite long and makes about 60% of the total execution time. Therefore we can not profit much from the use of special hardware (e.g. a neural network chip).

Another possibility is vectorization. But since we are using highly compressed vectors instead of full matrices we found only a small improvement (around 20% for our test sample).

5 Conclusion

We have developed a NN algorithm for track finding in a realistic environment. Due to a simplification of the connection matrix and some efficient cuts we obtain results of good quality with minimal memory and modest execution time demands. Although the calculation of the weights is quite simple the algorithm spends almost as much time for the initialization of the net as for the iterations. Therefore the use of NN chips is not very promissing for such applications. Also we have shown that this method can handle a quite big number of coordinates (> 4000) and tracks (200).

Acknoledgments

We would like to thank our colleagues of the Aleph collaboration for their support in preparing this publication.

References

[1] G.L. Shaw, G. Palm, Brain Theory (World Scientific, Singapur) 1988.

[2] B. Humpert, Comp. Phys. Comm. 58 (1990) 223,
P. Treleaven, M. Vellasco, Comp. Phys. Comm. 57 (1989) 543.

[3] D.E. Rumelhart et al., Nature 323 (1986) 533.

[4] B. Denby, Comp. Phys. Comm. 49 (1988) 429.

[5] B. Denby, S.L. Linn, Comp. Phys. Comm. 56 (1990) 293.

[6] C. Peterson, Nucl. Instr. Meth. A 279 (1989) 537.

[7] M. Mezard, G. Parisi, M.A. Virasoro, Spin Glass Theory and Beyond (World Scientific, Singapur) 1987.

[8] J.J. Hopfield, D.W. Tank, Biol. Cybernetics 52 (1985) 141.

[9] C. Peterson, J.R. Anderson, Complex Systems 1 (1987) 995.

[10] The Aleph Collaboration, Nucl. Inst. Meth., to be published (CERN-EP/90-23).

[11] M.E. Mermikides, internal note ALEPH-TPCDET 84-69 (1984).

Figure captions

- **Figure 1:** Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (X-Y projection).
- **Figure 2:** Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (R-Z projection).
- **Figure 3:** Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (X-Y projection).
- **Figure 4:** Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (R-Z projection).
- **Figure 5:** Execution time as a function of the number of tracks.
- **Figure 6:** Display of the generated lines for a Monte Carlo event with 100 tracks (X-Y projection).
- **Figure 7:** Display of the activated lines after convergence for a Monte Carlo event with 100 tracks (X-Y projection).

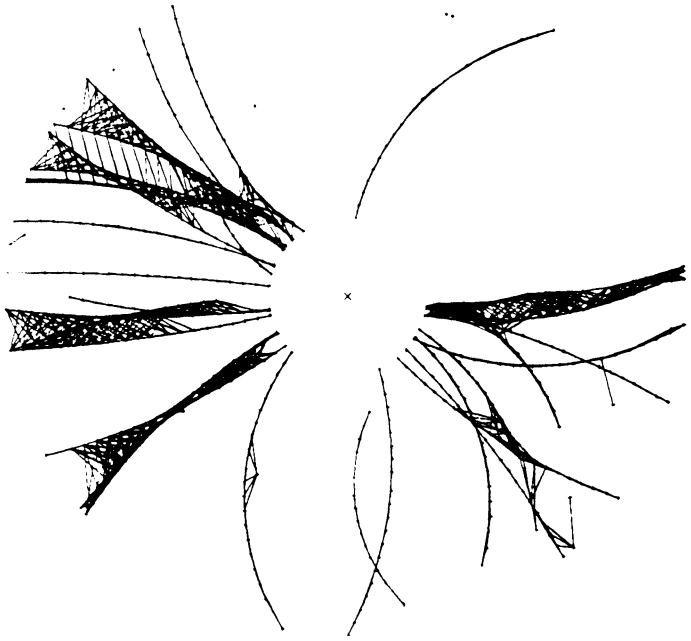

Figure 1: Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (X-Y projection).

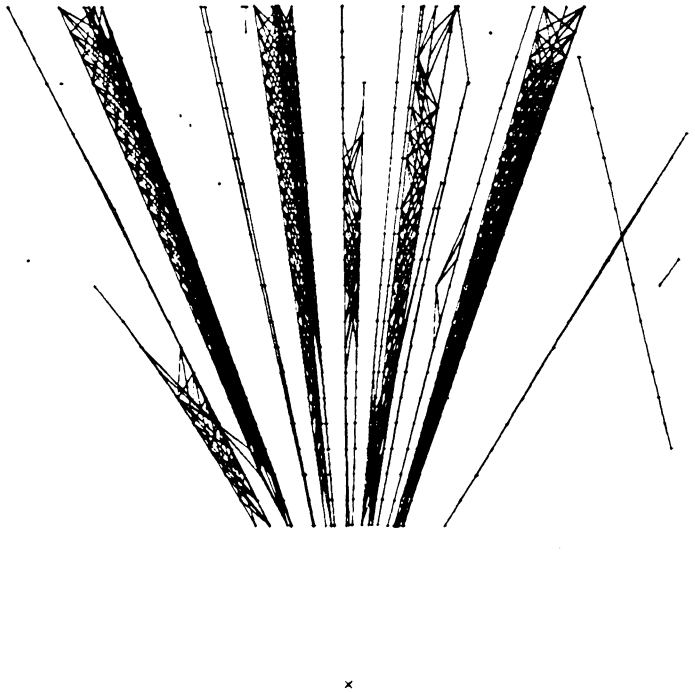

Figure 2: Display of all generated lines for a real $Z^0 \rightarrow$ hadrons (R-Z projection).

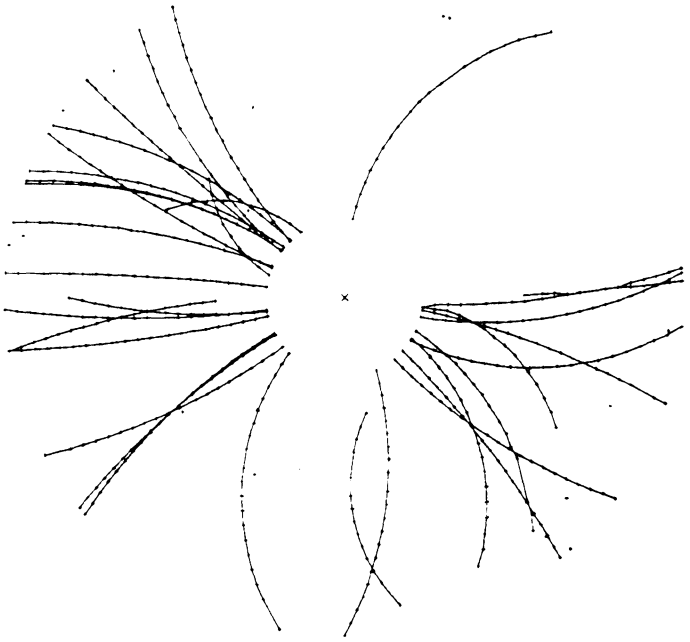

Figure 3: Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (X-Y projection).

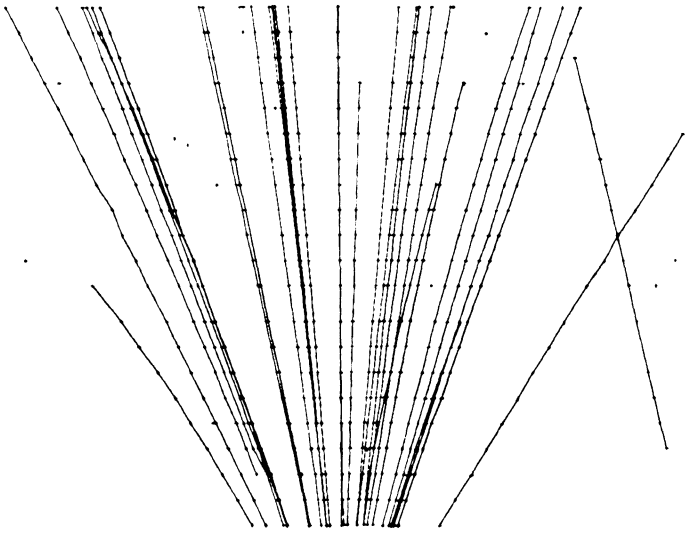

Figure 4: Display of the activated lines after convergence for a real $Z^0 \rightarrow$ hadrons (R-Z projection).

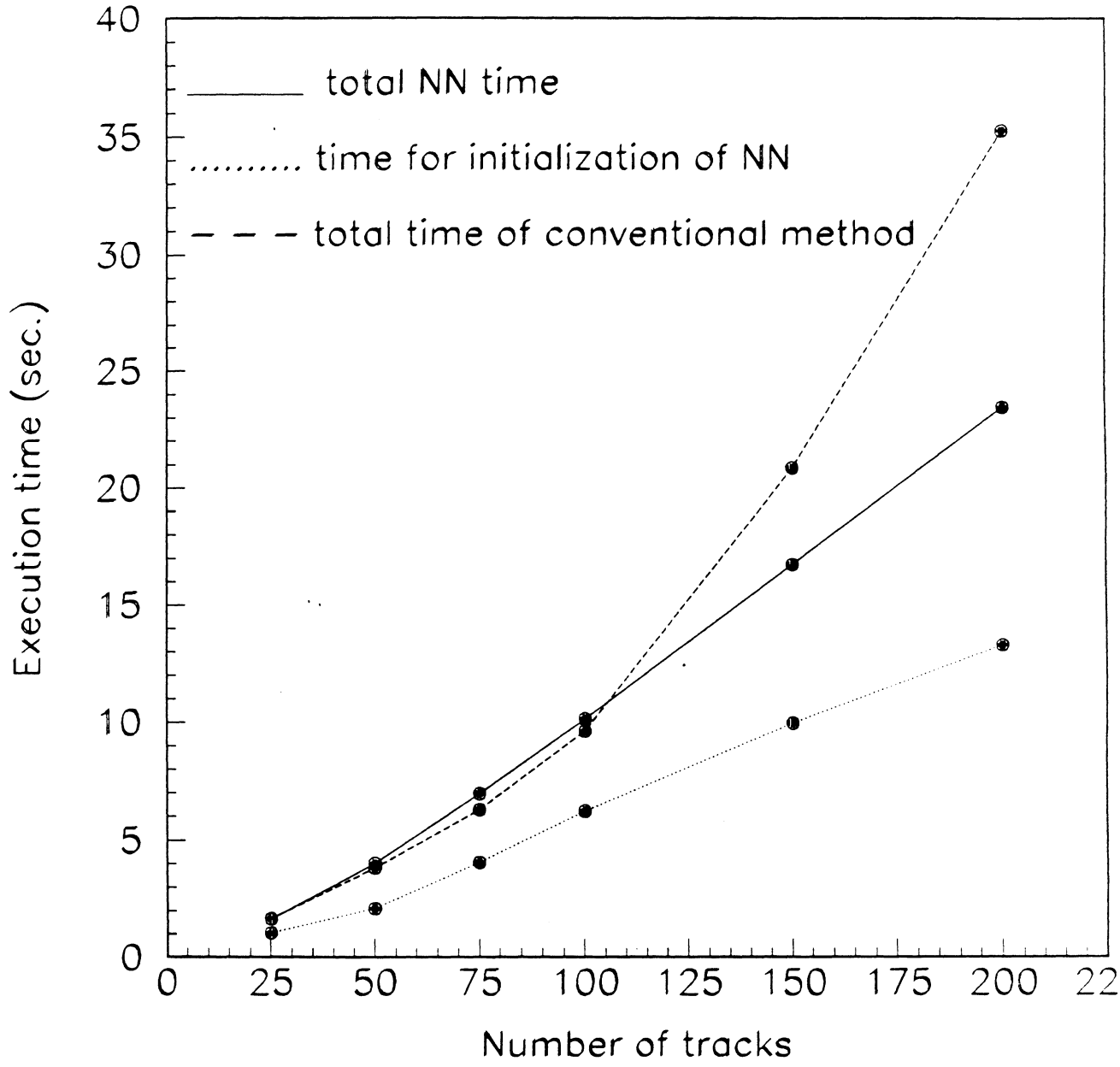

Figure 5: Execution time as a function of the number of tracks.

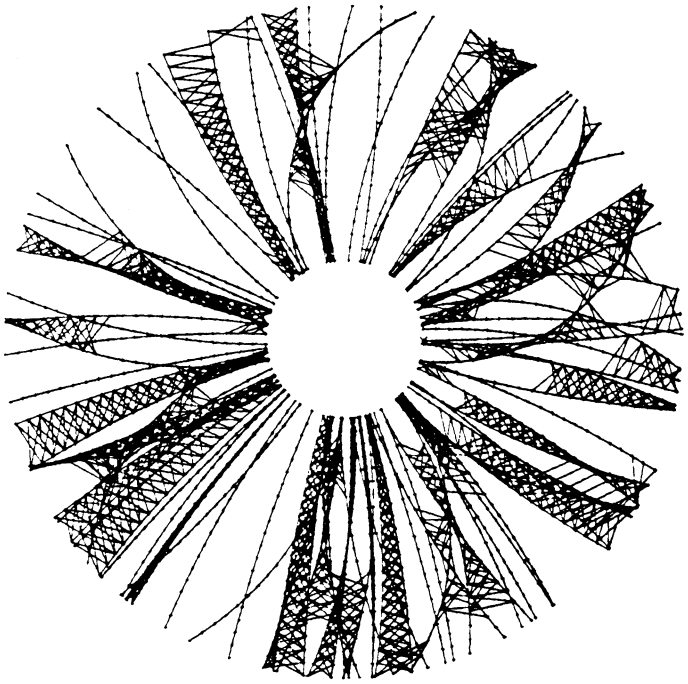

Figure 6: Display of the generated lines for a Monte Carlo event with 100 tracks (X-Y projection).

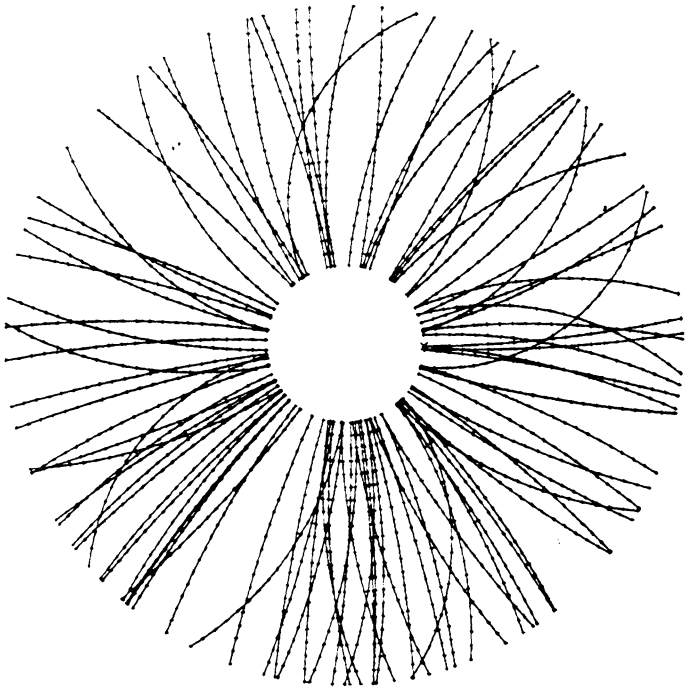

Figure 7: Display of the activated lines after convergence for a Monte Carlo event with 100 tracks (X-Y projection).