An algorithm of search of nearest neighbors leading to average computing time per point independent of the total number of points

Claude Delannoy

CENTRE NATIONAL D'ETUDES DES TELECOMMUNICATIONS

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

CENTRE DE RECHERCHES EN PHYSIQUE DE L'ENVIRONNEMENT TERRESTRE ET PLANETAIRE

CRPE

NOTE TECHNIQUE
CRPE / 36

An algorithm of search of nearest neighbors leading to average computing time per point independent of the total number of points

par

Claude DELANNOY

CENTRE NATIONAL D'ETUDES
DES TELECOMMUNICATIONS

CENTRE NATIONAL DE LA
RECHERCHE SCIENTIFIQUE

CENTRE DE RECHERCHE EN PHYSIQUE
DE L'ENVIRONNEMENT TERRESTRE ET PLANETAIRE

NOTE TECHNIQUE CRPE/ 36

AN ALGORITHM OF SEARCH OF NEAREST NEIGHBORS LEADING TO AVERAGE COMPUTING TIME PER POINT INDEPENDENT OF THE TOTAL NUMBER OF POINTS

par

Claude DELANNOY

C.R.P.E./P.C.E.
45045 - ORLEANS CEDEX, France

Le Chef du Département PCE

Le Directeur

Décembre 1976

A B S T R A C T

A new algorithm of search of nearest neighbors is proposed. It is based upon the partition of the working domain in cells. It presents two major characteristics. First, the average number of distance calculations can be done as small as required. Secondly, the average computing time per point is independant of the total number of points to be classified.

The theoretical behavior has been studied for a uniform distribution in a plan. Experimental verifications are provided.

1 - INTRODUCTION

Nearest neighbor techniques are important non parametric procedures used for multivariate density estimation and pattern classification. In both cases, we have to find the k nearest neighbors of some point x in a set E of N points ; in pattern classification x do not belong to E while it does in density estimation.

Many algorithms or preprocessing schemes have been proposed to reduce storage or processing requirements. But, in all cases, the computing time per point is an increasing function of N. Here, we are proposing a new algorithm which leads to average computing time per point independant of N.

In a first time, after problem formulation, we shall see the proposed method. Then we shall examine how such an algorithm theoretically behaves. Lastly experimental verifications will be described.

2 - PROPOSED METHOD

2.1. Problem Formulation

Let E and F be two sets of points of R^p (every point is defined by its p real coordinates)

We suppose that a distance d has been defined on R^p.

The problem is to find, for each point x belonging to F, what is called its "nearest neighbor in E" ; that means the point (supposed unique) $y_o \in E$ so that :

$$d(x,y_o) = \inf_{y \in E} d(x,y)$$

More generally, we could have to find the k nearest neighbors of each point of F. Here, for purpose of simplification, we shall suppose $k = 1$.

2.2. Basic idea

If we had to make a manual search of the nearest neighbor of a point x, we should proceed geometrically and we should consider only a neighborhood of x.

So the basic idea of the algorithm will be a simulation of this manual process. On that purpose we are going to make a partition of the whole domain G (union of the two sets : $G = E \cup F$) in cells.

So the search of y_o , nearest neighbor of x, will be limitated to a search in one or several cells.

We shall have two kinds of treatments :

(i) a preprocessing for the partition of G. This will be done once for all points of F.

(ii) a repetitive treatment which will be the same for each point of F.

Let us examine this in more details.

2.3. Preprocessing

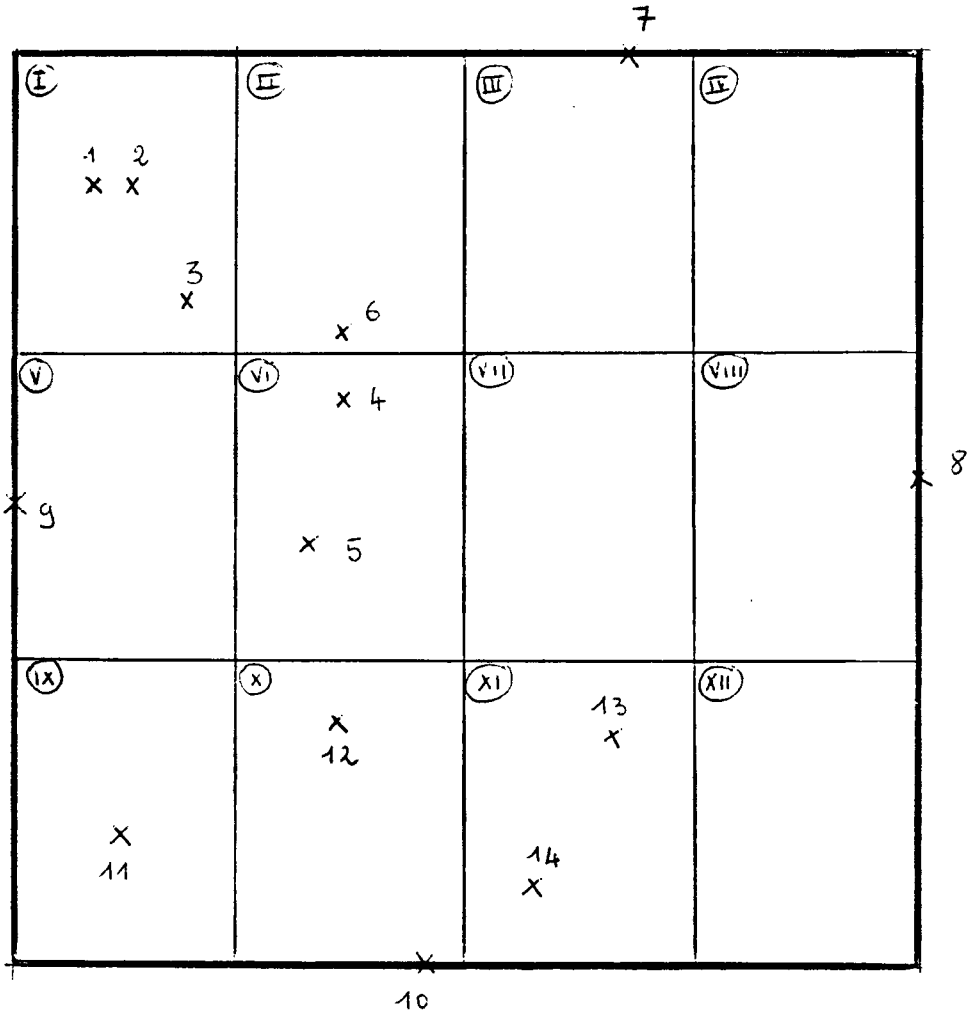
In a first time we determine the smallest hyper-cube of R^p (called H) which contains G. It is defined by p pairs of coordinates (min_i , max_i) $i = 1 , p$

Then we make a partition of each segment $[min_i , max_i]$ in λ_i parts. Let us note that we can choose as we want values of λ_i and the size of the parts.

In so doing , we have made a partition of H in $\prod_{i=1}^{p} \lambda_i$ cells. For each cell we set a list of points of E which belong to it.

2.4. Example

(i) Let us see how this preprocessing works on a simple example. Suppose we have E = F, 14 points of the plan (p = 2) distributed as shown in figure 1. Suppose that d is the euclidian distance. We choose $\lambda_1 = 4$, $\lambda_2 = 3$ and we make equal parts.



- Figure 1 -

In a first time, we determine the smallest rectangle containing our set G = E = F (in thick lines on fig. 1). Then we make a partition in 4 × 3 equal cells. So we can set the following list of points belonging to each cell (on figure 1 cells are identified by encircled numbers) :

cell	I	II	III	IV	V	VI	VII
points	1 2 3	6	7	-	9	4 5	-

and so on

Table 1 : List of points per cell

While we are considering this example, let us see how we may proceed to find the nearest neighbor of some point.

(ii) Search of the nearest neighbor of point 1

We determine the cell to which belongs point 1 ; it is cell number I. So, we first make a search in that cell. By looking up table 1, we have to compute two distances : d(1,2) and d(1,3). So we find that 2 is the nearest neighbor of 1 in cell I. Here d(1,3) is smaller than distance between point 1 and the outline of the cell I ; so we are sure that 2 is the nearest neighbor of point 1.

(ii) Suppose now that we have to find nearest neighbor of point 4.

Proceeding in the same way as for point 1, we find that point 5 is the nearest neighbor of 4 in cell VI. But d(4,5) is greater than distance between 4 and the outline of cell VI.

So we are not sure that 5 is really the nearest neighbor of 4 and it is necessary to extend the search to cells which are contiguous to cell VI. Here, we should have found that the nearest neighbor of 4 was point 6.

Let us see what the algorithm will be in a general case.

2.5. Algorithm of search of nearest neighbor of a point.

With following notations :

C_x^0 : cell containing point x

C_x^i : domain formed by the union of C_x^{i-1} and of the cells which are next to it.

(so C_x^i contains $(2\,i + 1)^p$ cells)

$d(x\ ,\ C_x^i)$: minimum distance between x and the outline of C_x^i,

the processing of one point can be done as in flow-chart of figure 2.

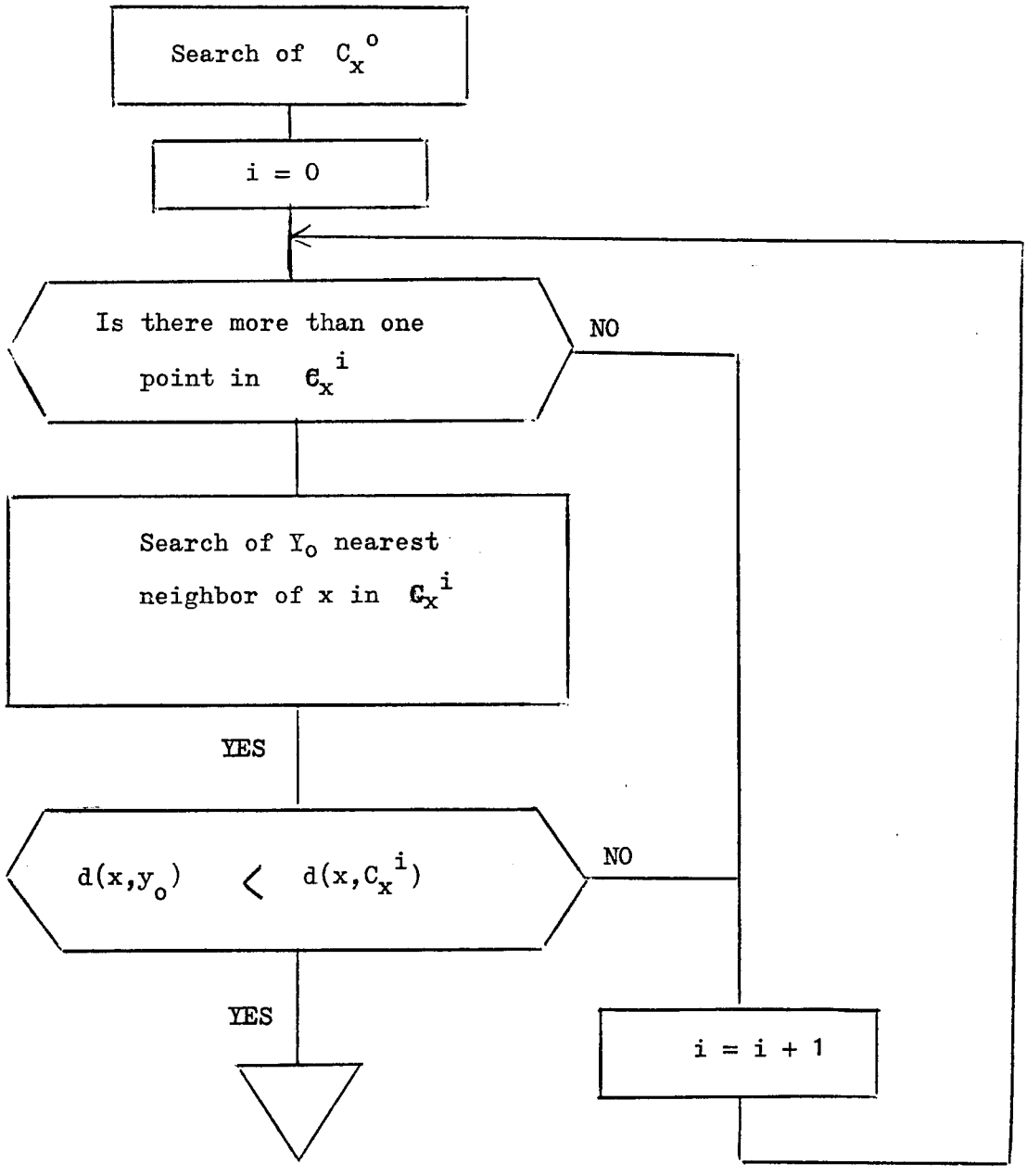


Figure 2 - Search of the nearest neighbor Y_o of one point x

3 - THEORETIC BEHAVIOUR OF THE ALGORITHM

3.1. Introduction

Usually such algorithms are characterized by the average number n_d of distances which it is necessary to compute (or to examine if we keep in memory an array of distances) to find the nearest neighbor of a point.

However the average computing time per point t can be expected to be about given by :

$$t = n_c t_c + n_d t_d \qquad (1)$$

where :
- n_c is the average number of cells that need to be examined to find the nearest neighbor of a point.
- t_c computing time of initialization of the search in a cell
- t_d time required by one distance calculation

So, we are going to examine how the quantities n_c and n_d behave and what conclusions may be applied to average computing time per point.

3.2. Hypothesis

Let us suppose that

(i) we are in a two dimensions problem ($p = 2$)
(ii) $E = F$ so that $G = E = F$
(iii) the N points of G are drawned according to a uniform distribution
(iv) we have N_c square cells of same size
(v) there are no edges effects

3.3. Expression of P_k

In a first time, let us see what are the expressions of quantities P_k : probability that the nearest neighbor of a point x belong to C_x^k (C_x^k being domain defined in section 2.5.)

As distribution of point x is uniform, P_k will be independant of x and we can write :

$$P_k = \int_o^\infty h(y) \; P(y) \; dy \qquad (2)$$

with $P(u)$: probability that a circle of ray u contain at least one point.

$h(u)$: distribution of the random variable : distance between some point and the edge of domain C_x^k

(i) Determination of P_k

The probability that in a circle of ray u, we find at least one point is :

$$P(u) = 1 - \left(1 - \frac{\pi u^2}{N_c}\right)^N \qquad (3)$$

To avoid edges effects, let us suppose that

$$N \to \infty, \quad N_c \to \infty \quad \text{with} \quad \frac{N}{N_c} = N_{pc} = \text{constant}$$

Then (3) becomes

$$P(u) = 1 - e^{-\pi N_{pc} u^2} \qquad (4)$$

(ii) Determination of h

The probability that distance d between a point x and the edge of C_x^k be smaller than k + d is given by

$$H(k + \alpha) = 1 - 4\,(\tfrac{1}{2} - \alpha)^2 \quad \text{for } \alpha \in \left[0\ ,\ \tfrac{1}{2}\right]$$

So, the distribution h is :

$$h(k + \alpha) = 4(1 - 2\alpha) \quad \text{if } \alpha \in \left[0\ ,\ \tfrac{1}{2}\right]$$

with $k + \alpha = v$, we find

$$\begin{aligned} h(v) &= 4(1 + 2k - 2v) \quad \text{if } v \in \left[k\ ,\ k + \tfrac{1}{2}\right] \\ h(v) &= 0 \qquad\qquad\quad \text{if } v \notin \left[k\ ,\ k + \tfrac{1}{2}\right] \end{aligned} \tag{5}$$

(iii) Determination of P_k

Using relations (4) and (5) in (2) we find

$$P_k = 4 \int_k^{k+\frac{1}{2}} (1 + 2k - 2x)\,(1 - e^{-\lambda^2 x^2})\,dx$$

with $\lambda^2 = \pi N_{pc}$

Finally we obtain :

$$P_k = 1 + \frac{4}{\lambda^2}\, e^{-k^2\lambda^2}\,(1 - e^{-\lambda^2(k+\frac{1}{4})}) - \frac{4(1+2k)\sqrt{\pi}}{\lambda}\, G(\lambda\sqrt{2}(k+\tfrac{1}{2}) - G(\lambda\sqrt{2}k) \tag{6}$$

with : $$G(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}}\,dt$$

We can remark that P_k is only depending on k and N_{pc}

Especially it is independant of N.

3.4. Behaviour of n_d and n_c

(i) The average number of cells that need be examined to find the nearest neighbor of some point is given by :

$$n_c = P_o + \sum_{k=1}^{\infty} 8\,k(P_k - P_{k-1}) \qquad (7)$$

where : 8 k represents the number of cells which belongs to C_x^k and not to C_x^{k-1}

$P_k - P_{k-1}$ is the probably that nearest neighbor of x be in C_x^k and not in C_x^{k-1}

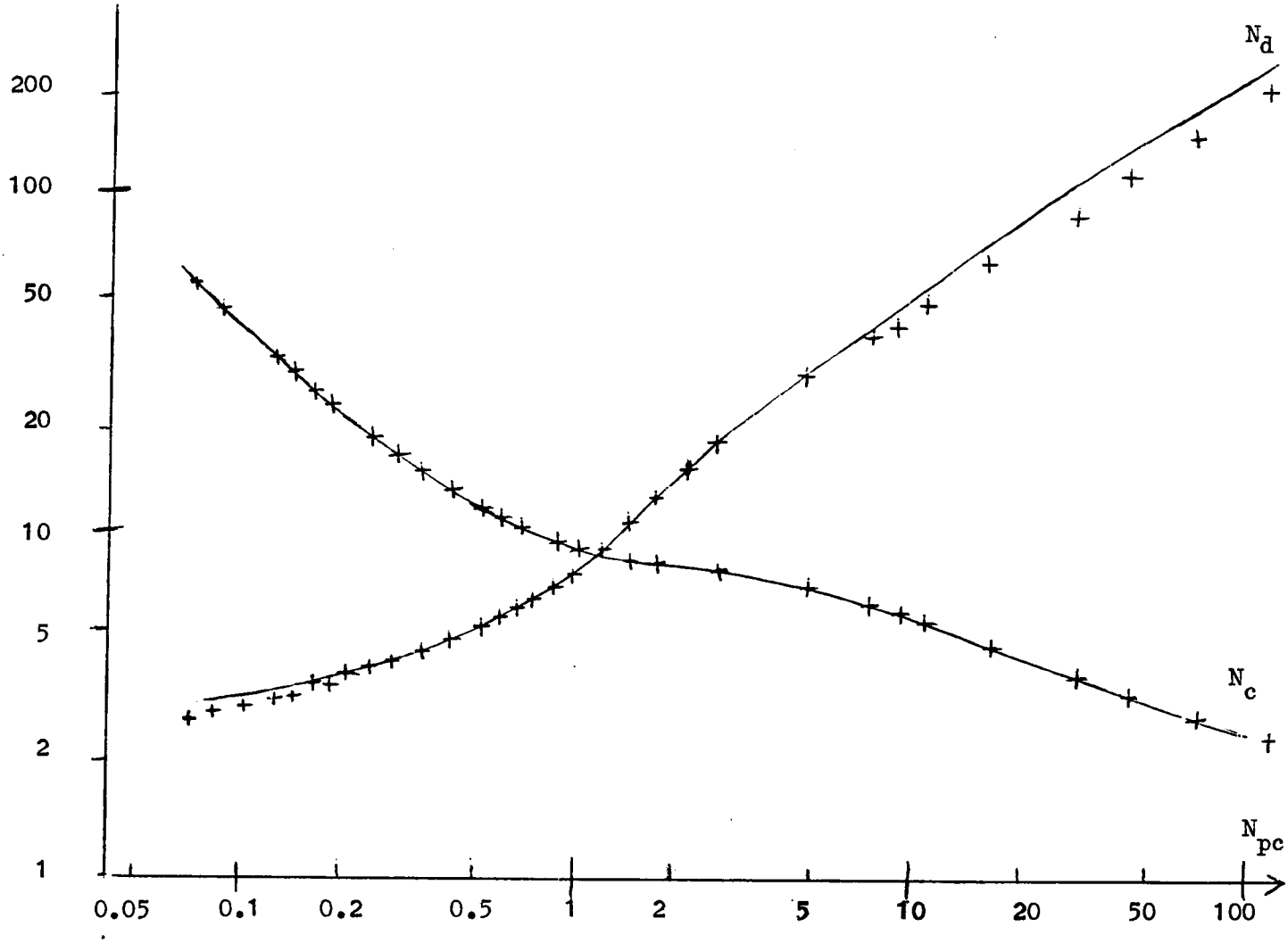
(ii) In the same way , the average number of distances that need be computed to find the nearest neighbor of a point is given by

$$n_d = P_o\ M_o + \sum_{k=1}^{\infty} (P_k - P_{k-1})\ M_k \qquad (8)$$

where M_k is the average number of neighbors of a point x belonging to C_x^k ; it may be shown to be given by

$$M_k = (2\,k + 1)^2\ N_{pc}\ (1 - e^{-(2\,k + 1)^2\ N_{pc}}) \qquad (9)$$

(iii) So, using (6) in (7) and (8), we can compute theoretical values of n_d and n_c in function of N_{pc}. They are represented in full lines on figure 3.



- Figure 3 -

Theoretical (full lines) and experimental (crosses) values of n_c and n_d in function of N_{pc}

3.5. Consequences

(i) the behaviour of the algorithm is independant of the total number N of points.

(ii) the number of distances calculations can be done as small as we want (within the limits $n_d > 1$). However, in practice we must consider average computing time per point t given by (1) in section 3.1. Figure 3 shows that t will present one or some minima for values of N_{pc} which will depend on value of $\frac{t_d}{t_c}$. So optima values of N_{pc} will be depending on programmation techniques used.

4 - EXPERIMENTAL VERIFICATIONS

4.1. Working hypothesis

To confirm theoretical behaviour, we proceeded so :

(i) a program was written in Fortran IV, it ran on IBM 370/168

(ii) E was a set of N points drawned at random according to a uniform distribution in the domain

$$[0 , 1] \times [0 , 1] \quad (\text{so} \quad p = 2)$$

(iii) We made a partition of the domain in λ^2 equal cells (so $\lambda_1 = \lambda_2 = \lambda$)

(iv) We searched the nearest neighbor in E of each point of E.

4.2. Behaviour of n_d and n_c in function of N_{pc}

For $N = 1000$, we computed average values n_d and n_c for different values of λ corresponding to different values of N_{pc}

Results are represented by crosses on figure 3.

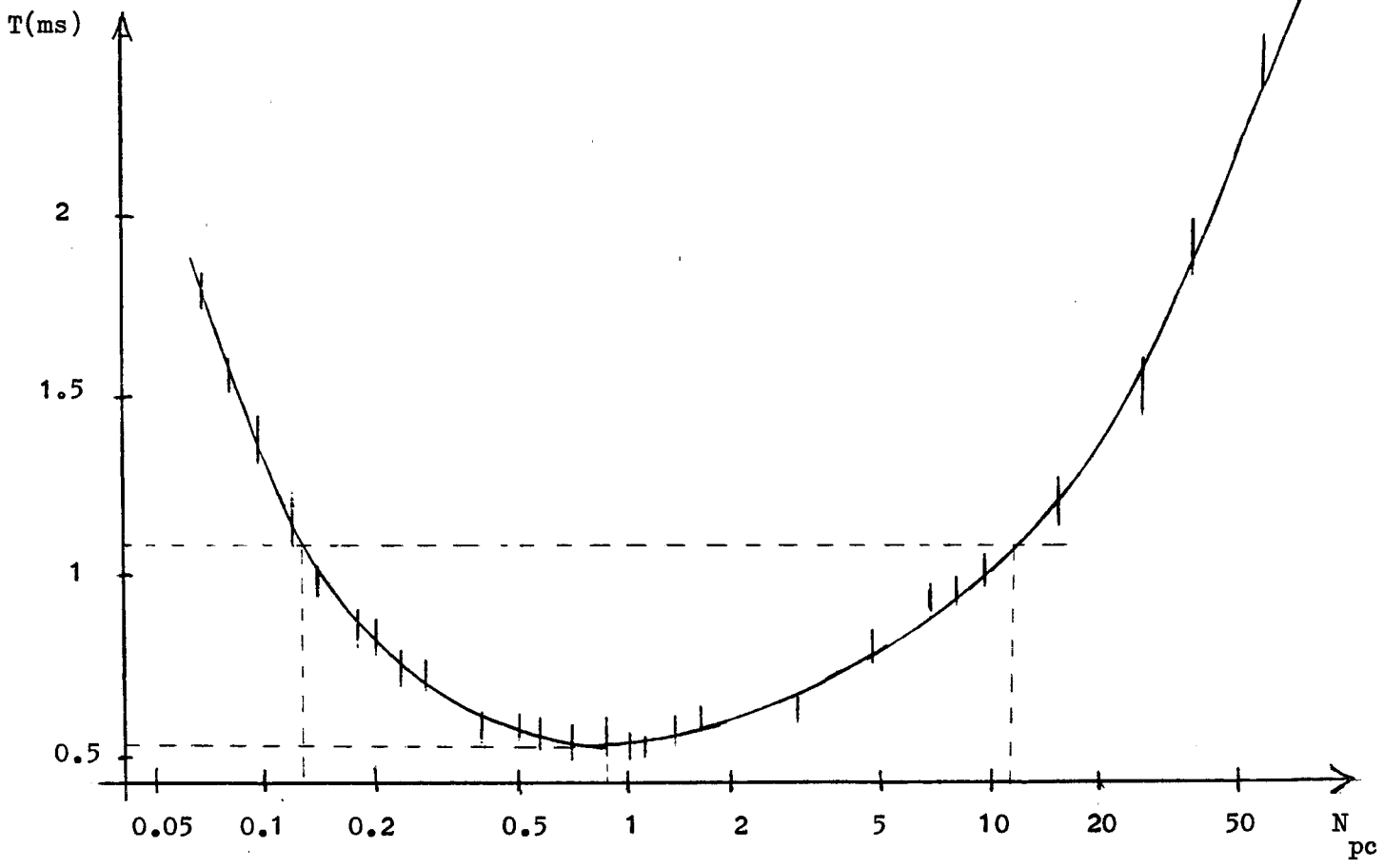
As far as n_c is concerned, we have an excellent verification of theory. For n_d , small divergences are appearing at the extremeties of the curve ; they can be explained by edges effects.

4.3. Average computing per point

In each of the preceding cases, average computing time per point was determined. It is shown in figure 4 (vertical segments represent variances of the errors on experimental measures).

We can remark that only one minimum is appearing.

In its neighbourhood, we note that computing time is multiplied by 2 when N_{pc} is about multiplied or divided by 10. This fact can be very useful in practical cases because generally N_{pc} will not be constant on the whole domain G



- Figure 4 -

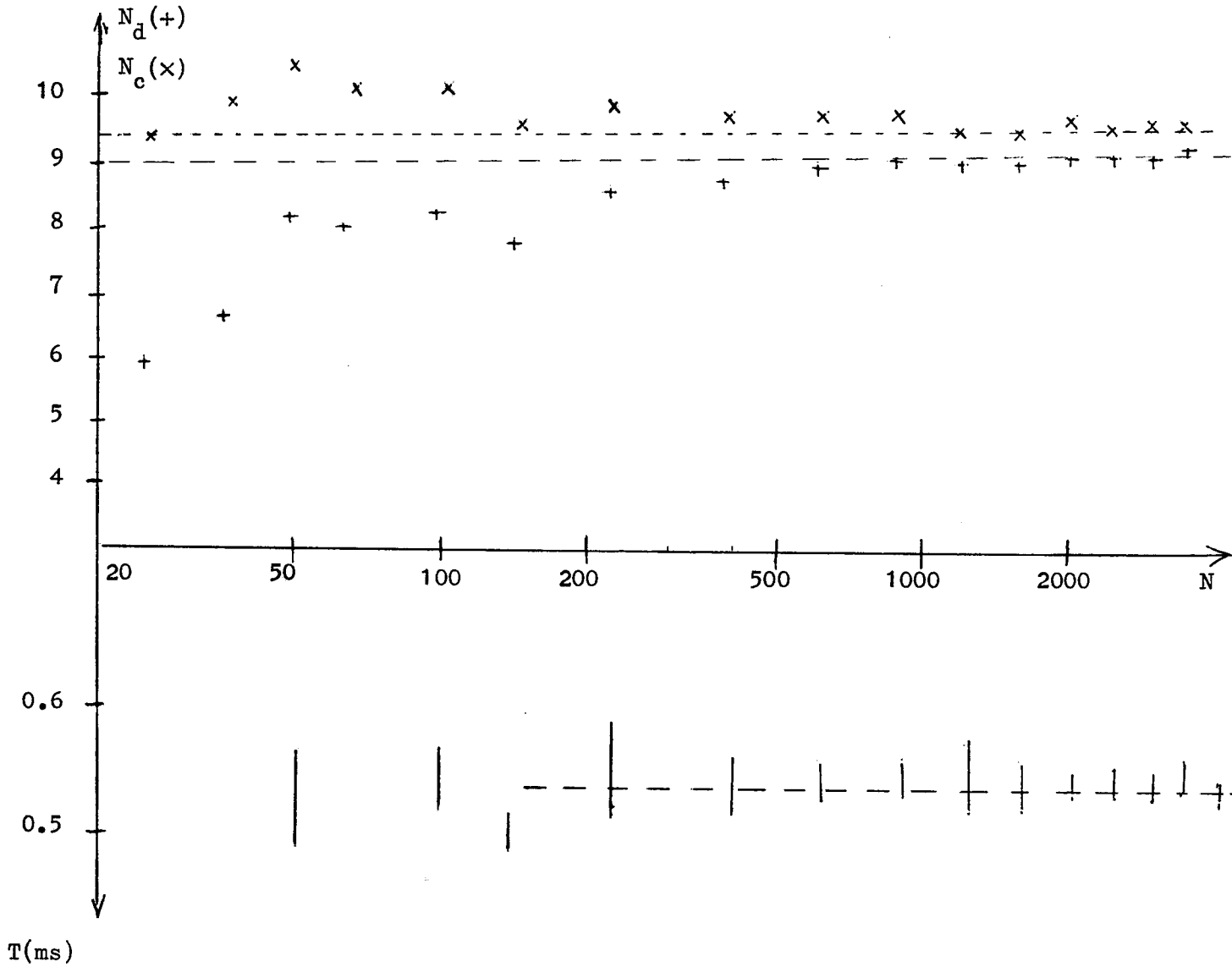
Average computing time per point

p = 2 N = 1000

4.4. Indepandance of N

For a given value of N_{pc} (1 here) we determined n_d , n_c and t for different values of N.

The results (figure 5) show that, for N sufficiently great (to avoid edges effects) experimental values of n_c and n_d (crosses) are very closed to the theoretical ones (in dotted lines). In the same time, value of t is significantly becoming constant.



- Figure 5 -

n_c , n_d and T in function of N

4.5. Remark

Other experimental determinations were made which are not presented here. Especially independance of N has been verified for

(i) uniform distribution for $p = 3, \ldots\ldots 8$

(ii) normal distribution for $p = 2$

5 - CONCLUSIONS

We think that more theoretical studies are necessary about this algorithm. Especially, distributions different of uniform one should be considered.

Moreover, on a practical point of view, computing time per point could be improved by some techniques. Among them, we can note :

(i) Suppression of some distance calculations

(ii) Lowering n_c by considering cells in a certain order

(iii) More efficient table look-up (writing in assembler langage or using hardware table look-up)

(iv) Cells of variable size ; the size could be determined in function of local density.

However, we may already note that this algorithm presents two major characteristics :

(i) The average number of distances calculations can be done as small as required.

(ii) The average computing time per point is independant of the number of points to be classified. So, in comparison with others algorithms, it will be as much better as we shall consider greater sets of points.

- R E F E R E N C E S -

1 Richard O. Duda, Peter E. Hart, Pattern classification and scene analysis, Willey interscience, 1973.

2 Jerome H. Friedman, Forest Baskett, Leonard J. Shustek, An algorithm for finding nearest neighbors, IEEE Trans. on computers, october 75.

3 Hart, P.E., The condensed nearest neighbor rule, IEEE Trans. Computers, May 68.

4 E.A. Patrick, F. Fischer, A generalized k nearest neighbor rule, Info and control 16, 128-152 (1970).

CRPE
Centre de Recherches
en Physique de l'Environnement
terrestre et planétaire

Avenue de la Recherche scientifique
45045 ORLEANS CEDEX

Département PCE
Physique et Chimie
de l'Environnement

Avenue de la Recherche scientifique
45045 ORLEANS CEDEX

Département ETE
Etudes par Télédétection
de l'Environnement

CNET - 38-40 rue du général Leclerc
92131 ISSY-LES-MOULINEAUX

CNET Imp 75/1163