CHAPTER 2 - FAST NEAREST NEIGHBOUR SEARCHING

2.1 INTRODUCTION

Given a set of n multidimensional vectors, the nearest neighbour (NN) problem is to find the closest vector to some given vector. Efficient NN-finding algorithms may be incorporated into clustering algorithms, as will be seen in Chapters 3 and 4. NN-finding algorithms can be made the basic "building blocks" of many commonly-used clustering algorithms, and thus progress in the area of NN-finding has immediate beneficial repercussions on clustering.

The NN problem is also important in its own right in other areas. In discriminant analysis in pattern recognition, a k-nearest neighbour assignment strategy is widely used : an unlabelled sample is assigned to the class to which a majority of its NNs belong (see Kittler, 1978). In database design, the NN problem is known as the best match problem : the closest record to a query is to be accessed efficiently (see Bentley, 1979). In information retrieval, a similar problem arises when the NN document of a query or request vector is required (see Perry and Willett, 1983). In such problems as minimizing head movement on direct access I/O devices, or the optimal sequencing of pens for plotting devices, a NN-based algorithm has been found to be both simple and effective (see Bentley, 1984).

Sections 2.2 and 2.3 both describe particular data structures where the objective is to break the O(n) barrier for determining the NN of a point, - in the average case if not in the worst case. These approaches have been very succeessful, but they are restricted to low dimensional NN-searching. For higher-dimensional data, a wide range of bounding approaches have been proposed, i.e. a theoretically smallest possible dissimilarity (the lower bound on the dissimilarity) is compared against the current NN dissimilarity, and if the former is the greater then there is no need to proceed to an exact calculation of the dissimilarity. Bounding approaches remain O(n) algorithms, but with a low constant of proportionality such algorithms can be very efficient.

Bounding approaches of a general nature are discussed in sections 2.4 and 2.5. The general principle of the branch and bound algorithm described in section 2.5 appears to be particularly versatile for general purpose applications. In section 2.6, the particular problem-area of information retrieval is focussed on. Finally, section 2.7 indicates what practical problems require attention at the present time.

2.2 Hashing

In this approach to NN-searching, a preprocessing stage precedes the searching stage. All points are mapped onto indexed cellular regions of space, so that NNs will be found in the same or in closely adjacent cells. Taking the plane as an example, and considering points (x_1, y_1), the maximum and minimum values on all coordinates are obtained (e.g. x_{min}, x_{max}). The mapping $x_i \rightarrow \lfloor (x_i - x_{min})/r \rfloor$, where constant r is chosen in terms of the number of equally spaced categories into which the interval $[x_{min}, x_{max})$ is to be divided, gives an integer value between 0 and $\lfloor (x_{max} - x_{min})/r \rfloor$ to x_i. Defining a similar mapping for y_i allows each point to be transformed onto a rectangular cell (see Fig. 2.1). O(nm) time is required to obtain the transformation of all n points in $\mathbb{R}^m$. The result is stored as a linked list with a pointer from each cell identifier to the set of all points which have been mapped onto that cell.

The implementation of NN searching proceeds as follows. Firstly, the closest point which is mapped onto the same grid cell as the target point is found. This will be a current NN point. A closer point may be mapped onto some other grid cell if the distance between target point and current NN point is greater than the distance between the target point and any of the boundaries of the cell containing it. If this is the case, all grid cells adjacent to the grid cell are searched in order to see if the current NN point can be bettered. It is, of course, possible that the initial cell might have contained no points other than the target point: in this case, the search through adjacent cells may yield a possible NN, and it may be necessary to widen the search to confirm this. Therefore in $\mathbb{R}^2$, 1 cell is searched first; if required, the 8 cells adjacent to this are searched; if necessary, the 16 cells adjacent to these cells (or less if restrained by the exterior walls of the region in which the search is taking place) are then searched; and so on.

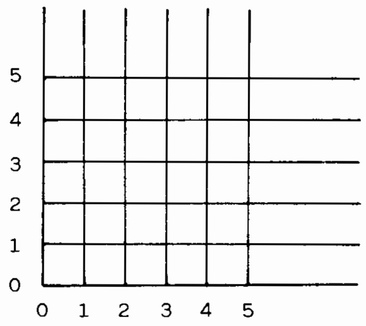
A powerful theoretical result regarding this approach is as follows: for uniformly distributed points the NN of a point is found in O(1) expected time (see Delannoy, 1980, or Bentley et al., 1980, for proof). Therefore this approach will work well if approximate uniformity can be assumed or if the data can be broken down into regions of approximately uniformly distributed points.

The search through adjacent cells requires time which increases exponentially with dimensionality (if it is assumed that the number of points assigned to each cell is approximately equal). As a result, this approach is suitable for low dimensions only. Rohlf (1978) reports on work in dimensions 2, 3 and 4, and Murtagh (1983) in $\mathbb{R}^2$. Rohlf also mentions the use of the first 3 principal components to approximate a set of points in 15-dimensional space.

$x_{min}, y_{min} = 0$

$x_{max}, y_{max} \leqslant 50$

$r = 10$



Point (21,40) is mapped onto cell (2,4)

Point (7,11) is mapped onto cell (0,1)

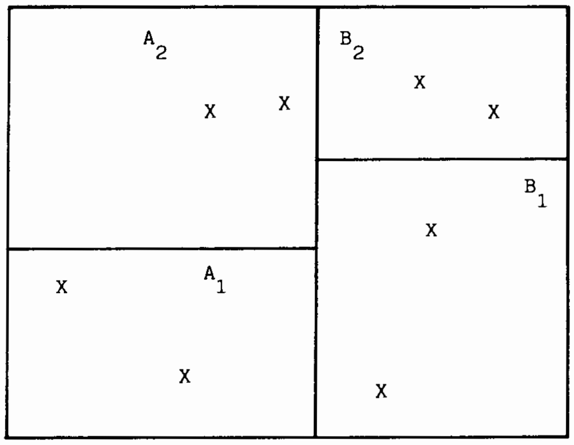
<u>Fig. 2.1</u> - Example of hashing in $\mathbb{R}^2$.

2.3 MULTIDIMENSIONAL BINARY SEARCH TREE

A decision tree splits the data to be searched through into 2 parts; each part is further subdivided; and subdivisions continue until some prespecified number of data points is arrived at. In practice, we associate with each node of the decision tree the definition of a subdivision of the data only, and we associate with each terminal node a pointer to the stored coordinates of the points (perhaps on direct access storage).

One version of the multidimensional binary search tree (MDBST) is as follows. Halve the set of points, using the median of the first coordinate values of the points. For each of the resulting sets of points, halve them using the median of the second coordinate values. Continue halving; use the medians of all coordinates in succession in order to define successive levels of the hierarchical decomposition; when all coordinates have been exhausted, recycle through the set of coordinates; halt when the number of points associated with the nodes at some level is smaller than or equal to a prespecified constant, c. See example in Fig. 2.2.
Clearly the tree is kept balanced : therefore there are O(log n) levels, at each of which O(n) processing is required. Hence the construction of the tree takes O(n log n) time.

The search for a NN then proceeds by a top-down traversal of the tree : the target point is transmitted through successive levels of the tree using the defined separation of the two child nodes at each node. On arrival at a terminal node, all associated points are examined and a current NN selected. The tree is then backtracked : if the points associated with any node could furnish a closer point, then subnodes must be checked out.



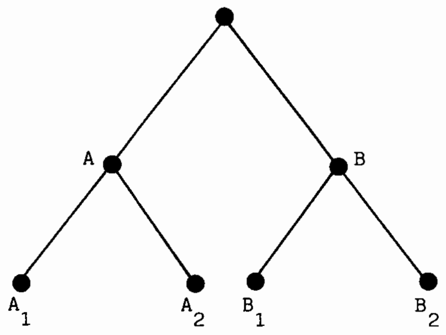


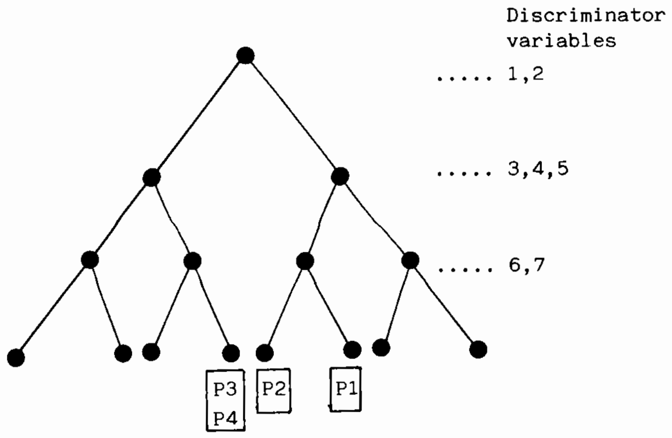
Fig. 2.2 - MDBST in $\mathbb{R}^2$.

The approximately constant number of points associated with terminal nodes (i.e. with hyper-rectangular cells in the space of points) should be greater than 1 in order that some NNs may be obtained without requiring a search of adjacent cells (i.e. of other terminal nodes). Friedman et al. (1977) suggest a value of c between 4 and 32 based on empirical study.

The MDBST approach only works well with small dimensions. To see this, consider each coordinate being used once only for the subdivision of points (i.e. each variable is considered equally useful). Let there be p levels in the tree, i.e. 2^p terminal nodes. Each terminal node contains approximately c points by construction and so $c.2^p = n$. Therefore $p = \log_2 n/c$. As sample values, if $n = 32768$, $c = 32$, then $p = 10$; i.e. in 10-dimensional space, using a large number of points associated with terminal nodes, more than 30000 points will need to be considered. For higher dimensional spaces, two alternative MDBST specifications are as follows.

All variables need not be considered for splitting the data if it is known that some are of greater interest than others. Linearity present in the data may manifest itself via the variance of the variables; choosing the variable with greatest variance as the discriminator variable at each node may therefore allow repeated use of certain variables. This has the added effect that the hyperrectangular cells into which the terminal nodes divide the space will be approximately cubical in shape. In this case, Friedman et al. (1977) show that search time is $O(\log n)$ on average for the finding of the NNs of all n points. Results obtained for dimensionalities of between 2 and 8 are reported on in Friedman et al. (1977), and in the application of this approach to minimal spanning tree (cf. Chapter 4) construction in Bentley and Friedman (1978).

In information retrieval, dimensionality is often greater than the number of points. Point coordinates here indicate the association or non-association (1 or 0) of an index term (variable) with a document (point). In this context, variables may be batched together. In Fig. 2.3 a target point with 0 coordinate values on attributes 1 and 2 is directed towards the left child node of the root; otherwise it is sent to the right child node; at level 2 if the target point has 0 coordinates for attributes 3, 4 and 5, it is directed towards the left subtree and otherwise towards the right subtree.
As with the version of the MDBST described above, a top-down traversal of the tree leads to the search of all points associated with a terminal node; this provides a current NN; then backtracking is commenced in order to see if the current NN can be bettered. The maximum size data set for which this approach has been used is 1400 documents. Results obtained have not been good (90% of documents required searching) but it is suggested that greater n would be more successful (Weiss, 1981; Eastman and Weiss, 1982). General guidelines for the variables which define the direction of search at each level are that they be related, and the number chosen should keep the tree balanced.



		Variables							
		1	2	3	4	5	6	7	8
Points	P1	1	0	0	0	0	1	1	1
	P2	1	1	0	0	0	0	0	0
	P3	0	0	1	0	0	1	1	0
	P4	0	0	0	1	1	1	0	0

Decision rule : presence of some one of discriminator variables

$\Rightarrow$ take right subtree.

Fig. 2.3 - MDBST using batching of variables for binary data.

2.4 BOUNDING USING PROJECTIONS OR THE EUCLIDEAN DISTANCE

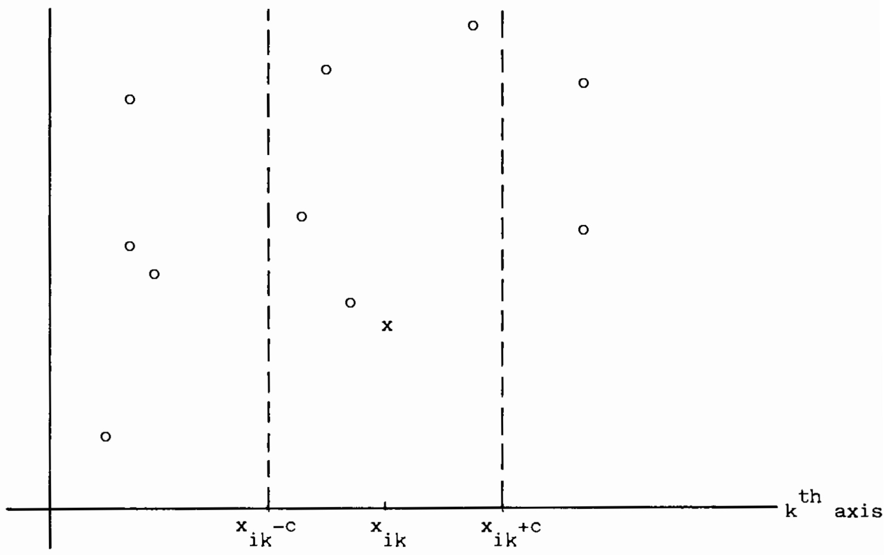
All remaining algorithms to be discussed make use of bounds : some lower bound on the dissimilarity is efficiently calculated in order to dispense with the full calculation in many instances.

Using projections on a coordinate axis allows the exclusion of points in the search for the NN of point x_i. Points x_j, only, are considered such that $(x_{ik} - x_{jk})^2 \leqslant c$, where $x_{ik}(x_{jk})$ is the k^{th} coordinate of x_i (x_j) and c is a prespecified constant (see Fig. 2.4).

Alternatively, more than one coordinate may be used. The prior sorting of coordinate values on the chosen axis or axes will expedite the finding of points whose full distance calculation is necessitated. The preprocessing required with this approach involves the sorting of up to m sets of coordinates, i.e. $O(mn \log n)$ time.

Using one axis, it is evident that many points may be excluded if the dimensionality is very small, but that the approach will disimprove as the latter grows. Friedman et al. (1975) give the expected NN search time, under the assumption that the points are uniformly distributed, as $O(m\, n^{1-1/n})$. This approaches the brute force $O(nm)$ as m gets large. Reported empirical results are for dimensions 2 to 8.

Marimont and Shapiro (1979) extend this approach by the use of projections in subspaces of dimension greater than 1 (usually about m/2 is suggested). This can be further improved if the subspace of the principal components (following a principal component analysis) is used. Dimensions up to 40 are examined.



Points with projections within distance c of given point's ("x") projection, alone, are searched.

Fig. 2.4 - Two-dimensional example of projection-based bound.

The Euclidean distance is the (dis)similarity which is most widely used in practice. Two other Minkowski distances require less computation time to calculate and provide bounds on the Euclidean distance. We have :

$$d_1(x,x') \geqslant d_2(x,x') \geqslant d_\infty(x,x')$$

where d_1 is the Hamming or "city block" distance, defined as $\sum_j |x_j - x'_j|$; the Euclidean distance is given by the square root of $\sum_j (x_j - x'_j)^2$; and the Chebyshev distance is defined as $\operatorname{Max}_j |x_j - x'_j|$.

Kittler (1978) makes use of the following bounding strategy: <u>reject all points y such that $d_1(x,y) \geqslant \sqrt{m}.\delta$</u> where δ is the current NN d_2-distance, and m is the dimensionality of the space. The more efficiently calculated d_1-distance may thus allow the rejection of many points (90% in 10-dimensional space in reported by Kittler). Kittler's rule is obtained by noting that the greatest d_1-distance between x and x' is attained when

$$|x_j - x'_j|^2 = \frac{d_2^2(x,x')}{m}$$

for all coordinates j; hence $d_1(x,x') = d_2(x,x')/\sqrt{m}$ is the greatest d_1-distance between x and x'. In the case of the rejection of point y, we then have :

$$d_1(x,y) \leqslant d_2(x,y)/\sqrt{m}$$

and since

$$d_1(x,y) \geqslant \sqrt{m}\,\delta$$

by virtue of the rejection, it follows that

$$\delta \leqslant d_2(x,y)$$

Yunck (1976) presents a theoretical analysis for the similar use of the Chebyshev metric. Richetin et al. (1980) propose the use of both bounds. Using uniformly distributed points in dimensions 2 to 5, the latter reference reports the best outcome when the rule : reject all y such that $d_\infty(x,y) \geqslant \delta$ precedes the rule based on the d_1-distance. Up to 80% reduction in CPU time is reported.

2.5 BOUNDING USING THE TRIANGULAR INEQUALITY

The triangular inequality is satisfied by distances : $d(x,y) \leqslant d(x,z)+d(z,y)$ where x,y and z are any three points. The use of a reference point, z, allows a full distance calculation between point x (whose NN is sought) and y to be avoided if

$$|d(y,z)-d(x,z)| \geqslant \delta$$

where δ is the current NN distance. The set of all distances to the reference point are calculated and stored in a preprocessing step requiring O(n) time and O(n) space. The above cut-off rule is obtained by noting that if

$$d(x,y) \geqslant |d(y,z)-d(x,z)|$$

then, necessarily, $d(x,y) \geqslant \delta$. The former inequality above reduces to the triangular inequality irrespective of which of d(y,z) or d(x,z) is the greater.

The set of distances to the reference point, $\{ d(x,z) | x \}$, may be sorted in the preprocessing stage. Since d(x,z) is fixed during the search for the NN of x, it follows that the cut-off rule will not then need to be applied in all cases.

The single reference point approach, due to Burkhard and Keller (1973), was generalized to multiple reference points by Shapiro (1977). The sorted list of distances to the first reference point, $\{d(x,z_1)|x\}$, is used as described above as a preliminary bound. Then the subsequent bounds are similarly employed to further reduce the points requiring a full distance calculation. The number and the choice of reference points to be used is dependent on the distributional characteristics of the data. Shapiro (1977) finds that reference points ought to the located away from groups of points.

In 10-dimensional simulations, it was found that at best only 20% of full distance calculations were required (although this was very dependent on the choice of reference points).

Fukunaga and Narendra (1975) make use of both a hierarchical decomposition of the data set (they employ repeatedly the k-means iterative/relocatory algorithm), and bounds based on the triangular inequality. For each node in the decomposition tree, the centre and maximum distance to the centre (the "radius") of associated points are determined. For 1000 points, the above reference uses 3 levels where there was a division into 3 classes at each node.

All points associated with a non-terminal node can be rejected in the search for the NN of point x if the following rule (Rule 1) is not verified:

$$d(x,g) - r_g < \delta$$

where δ is the current NN distance, g is the centre of the cluster of points associated with the node, and r_g is the radius of this cluster. For a terminal node, which cannot be rejected on the basis of this rule, each associated point, y, can be tested for rejection using the following rule :

$$| d(x,g)-d(y,g)| \geqslant \delta.$$

These two rules are direct consequences of the triangular inequality. A branch and bound algorithm may be implemented using these 2 rules. This involves determining some current NN (the bound) and subsequently "branching" out of a traversal path whenever the current NN cannot be bettered. The control structure for a backtracking implementation of this is shown in Fig. 2.5. The current NN distance will be initialized to machine infinity, and subsequently updated.

This NN searching approach appears particularly promising for general-purpose applications. It is also not inherently limited by dimensionality considerations.

```
node ← root.
call FUKNAR (node).
-
-
-
FUKNAR (node) :
if node = terminal then call TERM (node).
                    else if Rule-1-verified then node ← left-subnode;
                                                 call FUKNAR (node).
node ← next-subnode-from-the-left; call FUKNAR (node).
end
```

FUKNAR is a recursive procedure to implement backtracking NN search.

TERM is a procedure to carry out Rule 2 search of all points associated with the terminal node.

Rule-1-verified may be implemented by a procedure call.

The assignments to variable "node" require a choice of storage structure for the tree, and an associated set of flags to indicate when a node is not to be reused.

Fig. 2.5 - Control structure for backtracking NN searching.

2.6 NEAREST NEIGHBOUR ALGORITHMS IN INFORMATION RETRIEVAL

In information retrieval, m may typically be of the same order of magnitude as n or greater. Values in excess of 10,000 are not uncommon for test collections. The particular nature of the data (very sparse binary document-term associations) has given rise to particular NN-searching algorithms. The data is usually stored on direct access storage as compact linked lists (i.e. the sequence number of the document is followed by the sequence numbers of the terms which are associated with it). Current commercial document collections are usually searched using a Boolean search environment (i.e. all documents associated with particular terms are retrieved; the intersection/union/etc. of such sets of documents are obtained using AND/OR and other connectives). For efficiency, an _inverted file_ which maps terms onto documents is available for Boolean retrieval, - stored in a similar manner to the document-term file. Although storage considerations are clearly of paramount importance, it might _grosso modo_ be stated that we have available both the document-term matrix, and its transpose. The efficient NN-searching algorithms, to be discussed, will make use of both document-term and term-document files.

The usual algorithm for NN-searching considers each document in turn, calculates the dissimilarity with the given document (or query, which mathematically, is identical to a document), and updates the current NN if necessary. This algorithm is shown schematically in Fig. 2.6. The inner loop is simply an expression of the fact that the dissimilarity will (in general) require O(m) calculations: examples of commonly-used coefficients are the Jaccard similarity, i.e. $\# (i \cap i')/\# (i \cup i')$, and the Hamming distance, i.e. $\# (i) + \# (i') - 2 \# (i \cap i')$. Here, $\#$ is the counting operator, indicating the number of terms associated with document i(i.e. $\# (i)$) or the number of terms shared by documents i and i' (i.e. $\#(i \cap i')$).

Usual algorithm -

```
Initialize current NN.
For all documents in turn do :
... For all terms associated with the document do :
... ....... Determine dissimilarity
... End.
... Test against current NN.
End.
```

Croft's algorithm -

```
Initialize current NN.
For all terms associated with the given document do :
... For all documents associated with each term do :
... ....... For all terms associated with a document do :
... ....... ....... Determine dissimilarity.
... ....... End.
... ....... Test against current NN.
... End.
End.
```

Perry-Willett algorithm -

```
Initialize current NN.
For all terms associated with the given document (i) do :
... For all documents (i') associated with each term do :
... ....... Increment location i' of counter vector.
... End.
End.
```

Fig. 2.6 - Algorithms for NN-searching in information retrieval (see text for details).

If $\bar{m}$ and $\bar{n}$ are, respectively, the average numbers of terms associated with a document, and the average number of documents associated with a term, then the average complexity (over all n NN searches) of this usual algorithm is $O(n\ \bar{m})$. It is assumed that advantage is taken of some packed form of storage (i.e. some representation of linked lists) in the inner loop.

Croft's algorithm (see Croft ,1977) appears at first sight to be of complexity $O(n\ m^2)$. However the number of terms associated with the document whose NN is required will often be quite small. The National Physical Laboratory test collection (used in Smeaton and van Rijsbergen, 1981, and Murtagh, 1982), for example, has the following statistics : n = 11429, m = 7491, $\bar{m}$ = 19.9 and $\bar{n}$ = 30.4. The outermost and innermost loops in Croft's algorithm use the document-term file. The centre loop uses the term-document inverted file. The average complexity, over all n NN searches, is easily seen to be $O(\bar{n}\ \bar{m}^2)$.

In the outermost loop of Croft's algorithm, there will eventually come about a situation where - if a document has not been thus far examined - the number of terms remaining for the given document do not permit the current NN document to be bettered. In this case we can cut short the iterations of the outermost loop. The calculation of a bound, using the greatest possible number of terms which could be shared with a so-far unexamined document has been exploited by Smeaton and van Rijsbergen (1981) and by Murtagh (1982) in successive improvements on Croft's algorithm.

The complexity of all the above algorithms has been measured in terms of operations to be performed. In practice, however, the actual accessing of terms or documents is of far greater cost. The document-term and term-document files must be stored on direct access storage because of their large sizes. Carrying out an I/O operation is very much more costly than any operation in central memory. Note also that the strategy used in the foregoing algorithms (Croft's algorithm and the above-mentioned improvements on it) does not allow any viable approaches to batching together the records which are to be read successively, in order to improve the accessing performance.

The Perry-Willett algorithm (see Perry and Willett, 1983) presents a simple but effective solution to the problem of costly I/O. It focusses on the calculation of $\#(i \cap i')$, i.e. the number of terms common to the given document i and each other document i' in the collection. This set of values is built up in a computationally efficient fashion. O(n) operations are subsequently required to determine the (dis)similarity, using another vector, $\{ \#(i') | i' = 1,2,\ldots,n \}$, stored in main memory. Finally, the best (dis) similarity can be simultaneously determined when carrying out these operations. Hence computation time is $O(\bar{n}\ \bar{m} + n)$. We will now turn attention to the numbers of direct access reads required.

In Croft's algorithm, all terms associated with the document whose NN is desired may be read in one read operation. Subsequently, we require $\bar{n}\ \bar{m}$ reads, giving in all $1 + \bar{n}\ \bar{m}$. In the Perry-Willett algorithm, the outer loop again pertains to the one (given) document, and so all terms associated with this document can be read and stored. Subsequently, $\bar{m}$ reads (i.e. the average number of terms, each of which demands a read of a set of documents) are required, giving in all $1+\bar{m}$. Since these reads are very much the costliest operation in practice, the Perry-Willett algorithm can be recommended for large values of n and m. Its general limitations are that (1) it requires, as do all the algorithms discussed in this section, the availability of the inverted term-document file; and (2) it requires in-core storage of two vectors containing n integer values.

2.7 OPEN PROBLEMS

Approximation algorithms which find a close point, but which is not guaranteed to be a NN, have also been proposed. An exact search algorithm is however required for clustering (see Chapters 3 and 4) and in most other applications. More work is required in formulating general strategies for use with high-dimensional data : as may be noted in the previous sections, most experimentation has been on relatively small dimensions. For incorporation in clustering algorithms, the problem of dynamization is important also. This is where the NN algorithm caters for a dynamic data set, i.e. insertions and deletions to the data set are efficiently carried out.

2.8 REFERENCES.

J.L. BENTLEY, Multidimensional binary search trees in database applications. IEEE Transactions on Software Engineering SE-S, 333-340 (1979).

J.L. BENTLEY, A case study in applied algorithm design. Computer 17, 75-84 (1984).

J.L. BENTLEY and J.H. FRIEDMAN, Fast algorithms for constructing minimal spanning trees in coordinate spaces. IEEE Transactions on Computers C-27, 97-105 (1978).

J.L. BENTLEY, B.W. WEIDE and A.C. YAO, Optimal expected time algorithms for closest point problems. ACM Transactions on Mathematical Software 6, 563-580 (1980).

W.A. BURKHARD and R.M. KELLER, Some approaches to best-match file searching. Communications of the ACM 16, 230-236 (1973).

W.B. CROFT, Clustering large files of documents using the single-link method. Journal of the American Society for Information Science 28, 341-344 (1977).

C. DELANNOY, Un algorithme rapide de recherche de plus proches voisins. RAIRO Informatique/ Computer Science 14, 275-286 (1980).

C.M. EASTMAN and S.F. WEISS, Tree structures for high dimensionality nearest neighbour searching. Information Systems 7, 115-122 (1982).

J.H. FRIEDMAN, F. BASKETT and L.J. SHUSTEK, An algorithm for finding nearest neighbours. IEEE Transactions on Computers C-24, 1000-1006 (1975).

J.H. FRIEDMAN, J.L. BENTLEY and R.A. FINKEL, An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software 3, 209-226 (1977).

K. FUKUNAGA and P.M. NARENDRA , A branch and bound algorithm for computing k-nearest neighbours. IEEE Transactions on Computers C-24, 750-753 (1975).

J. KITTLER, A method for determinining k-nearest neighbours. Kybernetes 7, 313-315 (1978).

R.B. MARIMONT and M.B. SHAPIRO, Nearest neighbour searches and the curse of dimensionality . J. Inst. Maths. Applics. 24, 59-70 (1979).

F. MURTAGH, A very fast, exact nearest neighbour algorithm for use in information retrieval. Information Technology 1, 275-283 (1982).

F. MURTAGH, Expected time complexity results for hierarchic clustering algorithms which use cluster centres. Information Processing Letters 16, 237-241 (1983).

S.A. PERRY and P. WILLETT,A review of the use of inverted files for best match searching in information retrieval systems. Journal of Information Science 6, 59-66 (1983).

M. RICHETIN, G. RIVES and M. NARANJO, Algorithme rapide pour la détermination des k plus proches voisins. RAIRO Informatique/Computer Science 14, 369-378 (1980).

F.J. ROHLF, A probabilistic minimum spanning tree algorithm. Information Processing Letters 7, 44-48 (1978).

M. SHAPIRO, The choice of reference points in best-match file searching. Communications of the ACM 20, 339-343 (1977).

A.F. SMEATON and C.J. VAN RIJSBERGEN, The nearest neighbour problem in information retrieval : an algorithm using upperbounds. ACM SIGIR Forum 16, 83-87 (1981).

S.F. WEISS, A probabilistic algorithm for nearest neighbour searching. In R.N. Oddy et al. (eds.), Information Retrieval Research, Butterworths, London, 325-333 (1981).

T.P. YUNCK, A technique to identify nearest neighbours. IEEE Transactions on Systems, Man, and Cybernetics SMC-6, 678-683 (1976).