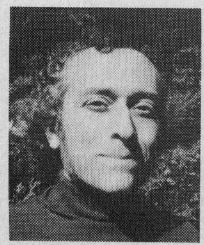
survey," *IEEE Trans. Audio Electroacoust.* (*Special Issue on Fast Fourier Transform*), vol. AU-17, pp. 108–119, June 1969.
[8] R. H. Fuller, "Associative parallel processing," in *1967 Spring Joint Computer Conf., AFIPS Conf. Proc.*, vol. 30. Washington, D. C.: Spartan, 1967, pp. 471–475.
[9] L. R. Bahl and H. Kobayashi, "Image data compression by predictive coding," *IBM J. Res. Develop.*, vol. 18, pp. 164–179, Mar. 1974.



R. N. Ascher was born in New York, N. Y., in 1923. He received the B.A. and M.A. degrees from Columbia University, New York, N. Y.

Following some years of graduate study in mathematics at Columbia, he spent eight years in Louisiana performing executive functions in an oil and gas producing company. In 1959 he joined the Scientific Computations Laboratory at the IBM Development Laboratory, Endicott, N. Y. Since 1967 he has been on the staff at the IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y.

George Nagy (SM'73) was born in Budapest, Hungary, in 1937. After early schooling in Hungary, Italy, France, and Canada, he obtained the B.S. degree in engineering physics and the M.S. degree in electrical engineering, both from McGill University, in 1959 and 1960, respectively, and the Ph.D. in electrical engineering from Cornell University, Ithaca, N. Y., in 1962.

Between 1963 and 1972 he was a Staff Member of the IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., conducting experiments in character recognition and, more recently, in remote sensing. In addition to several survey papers in the general area of pattern recognition, he has published articles on adaptive devices, analog matrix multipliers, image processing, feature extraction, the classification of Chinese ideographs, the use of context in reading machines, nonsupervised adaptive schemes in character recognition, and multispectral data processing. He is currently Professor and Chairman of the Department of Computer Science, University of Nebraska, Lincoln.

Finding Prototypes for Nearest Neighbor Classifiers

CHIN-LIANG CHANG, MEMBER, IEEE

***Abstract*—A nearest neighbor classifier is one which assigns a pattern to the class of the nearest prototype. An algorithm is given to find prototypes for a nearest neighbor classifier. The idea is to start with every sample in a training set as a prototype, and then successively merge any two nearest prototypes of the same class so long as the recognition rate is not downgraded. The algorithm is very effective. For example, when it was applied to a training set of 514 cases of liver disease, only 34 prototypes were found necessary to achieve the same recognition rate as the one using the 514 samples of the training set as prototypes. Furthermore, the number of prototypes in the algorithm need not be specified beforehand.**

***Index Terms*—Discriminant functions, generation of prototypes, minimal spanning tree algorithm, nearest neighbor classifiers, pattern recognition, piecewise linear classifiers, recognition rates, test sets, training sets.**

I. INTRODUCTION

ASSUME that an n-dimensional vector in a Euclidean space is a pattern. Let us also assume that there are r possible classes. The problem of designing a classifier for pattern recognition can be stated as follows: find r functions, $g_1,\cdots,g_r$, such that a pattern x is in class i if $g_i(x)$ is the optimal value among $g_1(x),\cdots,g_r(x)$. Each of these $g_1,\cdots,g_r$ is called a *discriminant function* [6]. There are many types of discriminant functions. In this paper, we shall consider classifiers based on nearest neighbor discriminant functions described below.

For $i = 1,\cdots,r$, let $p_i^1,\cdots,p_i^{k_i}$ be vectors in an n-dimensional Euclidean space E^n. If a discriminant function g_i is of the form

$$g_i(x) = \min \{d(x,p_i^1),\cdots,d(x,p_i^{k_i})\} \tag{1}$$

where $d(x,p_i^j)$ is a distance between x and p_i^j, g_i is called a *nearest neighbor discriminant function*. Note that $p_i^1,\cdots,p_i^{k_i}$ are often called *prototypes* (*reference points*) for class i. A classifier based on a set of nearest neighbor discriminant functions is called a *nearest neighbor classifier* [2], [5]. A nearest neighbor classifier assigns an unknown pattern to the class of the closest prototype. That is, a pattern x is assigned to class i if $g_i(x)$ is the smallest value among $g_1(x),\cdots,g_r(x)$. Although in a nearest neighbor classifier any distance measurement can be used, we shall restrict ourselves to the Euclidean distance.

In the sequel, for a pattern x, we shall use class (x)

Manuscript received November 29, 1973; revised March 4, 1974.
C.-L. Chang was with the Heuristics Laboratory, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Md. 20014. He is now with IBM Research Laboratory, San Jose, Calif. 95193.

to denote the class that x belongs to. Given a set T of sample patterns whose classes are known, our task is to design a nearest neighbor classifier which can classify most of the patterns in T correctly. T is usually called a *training set*. To accomplish this task, we need to determine not only the number k_i of prototypes but also prototypes $p_i^1,\cdots,p_i^{k_i}$ for each $i = 1,\cdots,r$. Obviously, the simplest way to obtain a nearest neighbor classifier is to use all points in the training set T as prototypes, and to assign an unknown pattern to the class of the closest point in T. A nearest neighbor classifier designed in this fashion achieves the highest recognition rate possible for the training set. However, this nearest neighbor classifier has one major drawback. That is, to classify an unknown pattern x, it requires the computation of distances between x and all points in the training set. In practice, since a training set T must contain all possible variations of patterns, T is usually very large. For example, in character recognition, it is not surprising that a training set contains thousands of sample patterns. Therefore, while maintaining the highest possible recognition rate, we would like to use a small number of prototypes. In this paper, we shall present a method to solve this problem.

II. AN ALGORITHM FOR DESIGNING A NEAREST NEIGHBOR CLASSIFIER

Suppose a training set T is given as $T = \{t^1,\cdots,t^m\}$. The idea of our algorithm is as follows: we start with every point in T as a prototype. We then successively merge any two closest prototypes p^1 and p^2 of the same class (i.e., replace p^1 and p^2 by a new prototype p) if the merging will not downgrade the classification of patterns in T. The new prototype p may be simply the average vector of p^1 and p^2, or the average vector of weighted p^1 and p^2. The class of the new prototype is the same as the one of p^1 and p^2. We continue the merging process until the number of incorrect classifications of patterns in T starts to increase. We give a simple example to illustrate the above idea. Suppose we are given a training set of samples shown in Fig. 1(a). We start with prototypes shown in Fig. 1(b) which is the same as Fig. 1(a). Note that a nearest neighbor classifier using the prototypes of Fig. 1(b) can correctly classify all patterns in Fig. 1(a). Now, since prototypes A and B are the closest and are of the same class, we try to merge them. If A and B are replaced by a new prototype H, all patterns in Fig. 1(a) are still correctly classified. Therefore, replacing A and B by H, we obtain a new set of prototypes shown in Fig. 1(c). Similarly, replacing H and C by I, we obtain Fig. 1(d). Merging F and G to J, we obtain Fig. 1(e). Finally, replacing D and E by K, we obtain Fig. 1(f). Using prototypes shown in Fig. 1(f), every pattern in Fig. 1(a) will still be correctly classified. However, if we continue to merge I and J, some patterns in Fig. 1(a) will be incorrectly classified. Therefore, we stop the merging process and the points shown in Fig. 1(f) will be used as the prototypes in a nearest neighbor classifier.

We now give an efficient algorithm to carry out the

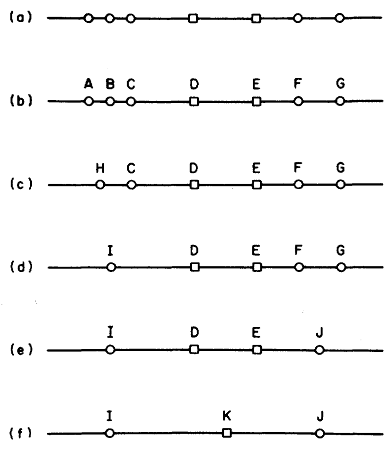


Fig. 1.

merging process. This algorithm is similar to the minimal spanning tree algorithm of Prim [7]. The minimal spanning tree algorithm is also used in the related problem of cluster analysis [9]. Our algorithm is specially tailored for pattern recognition, and has to deal with the generation of new prototypes and the associated problems. Roughly, our algorithm can be stated as follows:

Given a training set T, let initial prototypes be just the points of T. At any stage the prototypes belong to one of two sets—set A or set B. Initially, A is empty and B is equal to T. We start with an arbitrary point in B and initially assign it to A. Find a point p in A and a point q in B such that the distance between p and q is the shortest among all distances between points of A and points of B. Try to merge p and q. That is, if p and q are of the same class, compute a vector p^* in terms of p and q. If replacing p and q by p^* does not decrease the recognition rate for T, merging is successful. In this case, delete p and q from A and B, respectively, and put p^* into A, and the procedure is repeated again. In the case that p and q can not be merged, i.e., if either p and q are not of the same class or merging is unsuccessful, move q from B to A, and the procedure is repeated. When B becomes empty, recycle the whole procedure by letting B be the final A obtained from the previous cycle, and by resetting A to the empty set. This recycling is stopped when no new merged prototypes are obtained. The final prototypes in A are then used in a nearest neighbor classifier.

The above procedure W^* is just an outline. Some important parts of W^* will be discussed in detail as follows:

1) In procedure W^*, we have to compute p^* in terms of p and q. There are several ways to compute p^*. In this paper, we define p^* to be the average of weighted p and q. That is, we first let every initial prototype be associated with 1. If p and q are associated with integers M and N, respectively, p^* is defined as $p^* = (Mp + Nq)/(M + N)$, and is associated with the integer $(M + N)$. Note that p^* is the average of all initial prototypes contributed to p and q.

2) In procedure W^*, we need to find a point p in set A and a point q in set B such that the distance between p and q is the shortest among all distances between points

of A and points of B. An efficient way to find such a pair of points p and q is to use an algorithm similar to the minimal spanning tree algorithm given by Prim [7] and implemented by the program of Ross [8]. The idea is to store the distances between all points of B and their respective nearest points in A. Every time a new point is put into A, or A is changed, these distances are updated. Thus, from these distances, it is very fast to find two nearest points p and q such that p is in A and q is in B.

3) Once a pair of nearest points p and q is found, where $p \in A$ and $q \in B$, we need to test whether or not we can merge p and q. In this paper, we give an efficient method to perform this. The idea of our method is to associate with every point t^i in the training set $T = \{t^1,\cdots,t^m\}$ two distances w_i and b_i, where

w_i	distance between t^i and the nearest prototype of the same class as the class of t^i;
b_i	distance between t^i and the nearest prototype of the different class from the class of t^i.

The initial values of $w_1,\cdots,w_m$ and $b_1,\cdots,b_m$ can be obtained by the method described in the next paragraph. We first discuss how these values can be updated. Suppose p and q belong to class k, i.e., class (p) = class $(q) = k$. If p and q are merged to p^*, only some of $w_1,\cdots,w_m$ and $b_1,\cdots,b_m$ need to be updated. For $i = 1,\cdots,m$, w_i should be updated only if class $(t^i) = k$. In this case, if neither p nor q is the nearest prototype to t^i, w_i is updated to be $d(t^i,p^*)$ if $d(t^i,p^*)$ is smaller than the present w_i, and is unchanged otherwise. If p or q is the nearest prototype to t^i, let w_i be the smallest distance among all distances between t^i and prototypes different from p and q which are of the same class of t^i. On the other hand, for $i = 1,\cdots,m$, b_i should be updated only if class $(t^i) \neq k$. In this case, if neither p nor q is the nearest prototype to t^i, b_i is updated to be $d(t^i,p^*)$ if $d(t^i,p^*)$ is smaller than the present b_i, and is unchanged otherwise. If p or q is the nearest prototype to t^i, let b_i be the smallest distance among all distances between t^i and prototypes different from p and q which are of a different class from t^i. We note that for a pattern t^i in T to be correctly classified, w_i must be less than b_i. To test whether or not p and q can be merged to p^*, we try to use the above method to update $w_1,\cdots,w_m$ and $b_1,\cdots,b_m$ to, say, $w_1',\cdots,w_m'$ and $b_1',\cdots,b_m'$, respectively. If there exists such a condition that $w_i < b_i$ and $w_i' \geq b_i'$, i.e., t^i is correctly classified by the present set of prototypes but incorrectly classified by the would-be new set of prototypes, then p and q can not be merged. Otherwise, merging is to be performed.

Now, we describe how the initial values of $w_1,\cdots,w_m$ and $b_1,\cdots,b_m$ can be calculated as follows.

a) At the beginning of algorithm W^*, the prototypes are just points in the training set T. Therefore, since the nearest prototype to t^i of T is t^i itself, $w_i = 0$ for $i = 1,\cdots,m$.

b) Initially, $t^1,\cdots,t^m$ are both samples and prototypes. In this case, $b_1,\cdots,b_m$ can be efficiently calculated by an algorithm which is a modified version of the minimal spanning tree algorithm. That is, at any stage, $t^1,\cdots,t^m$ belong to one of two sets—set A^* and set B^*. Initially, A^* is empty and B^* is T. Also, initially set $b_i = \infty$ for $i = 1,\cdots,m$. Then the following steps are taken.

Step 1: Start with an arbitrary point t^j in B^* and assign it to A^*.

Step 2: For all points t^k in B^* such that class $(t^k) \neq$ class (t^j), update b_k to be the distance $d(t^k,t^j)$ between t^k and t^j if this distance is smaller than the present b_k. Otherwise, b_k is unchanged.

Step 3: Among all points in B^*, find a point t^s which has the smallest b_s associated with it.

Step 4: If t^j is not the nearest point to t^s such that the classes of t^j and t^s are different, go to Step 6. Otherwise, continue.

Step 5: Check whether or not $d(t^j,t^s)$ is less than b_j. If no, go to Step 6. If yes, let $b_j = d(t^j,t^s)$ and continue.

Step 6: Let $j = s$, move t^s from B^* to A^*, and go to Step 2 until B^* is empty. When B^* is empty, the final $b_1,\cdots,b_m$ are the desired ones.

Example: Consider the sample points shown in Fig. 2(a), where points t^1 and t^2 are in class 1, t^3 and t^4 class 2, and t^5 and t^6 class 3. Let these points also be prototypes. At the beginning, A^* is empty and $B^* = \{t^1,\cdots,t^6\}$. Suppose, in Step 1 of the above algorithm, we start with t^1 and initially assign it to A^*. After going through Steps 1 to 2, we should obtain $b_1,\cdots,b_6$ shown in Fig. 2(b), where b_i is indicated by a distance shown by an arc leaving from t^i and entering some nearest point (of the different class) so far found by the algorithm. If no arc is leaving t^i, that means $b_i = \infty$, $i = 1,\cdots,6$. Note that points assigned to A^* are indicated by check marks. In Fig. 2(b), since b_3 is the smallest, t^3 is assigned to A^*, and Steps 2–6 are repeated. Thus we obtain $b_1,\cdots,b_6$ shown in Fig. 2(c). Now, b_2 is the smallest. Therefore, t^2 is assigned to A^* and Steps 2–6 are repeated again. This is repeated again and again until B^* is empty. Fig.

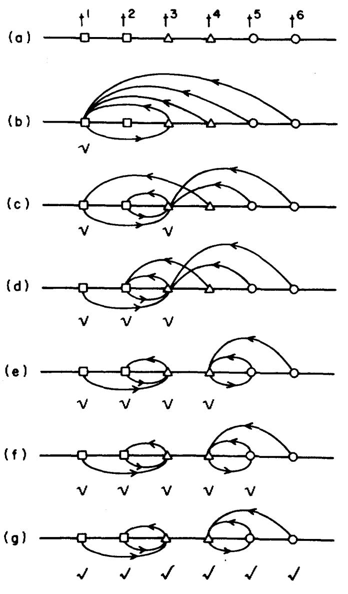


Fig. 2.

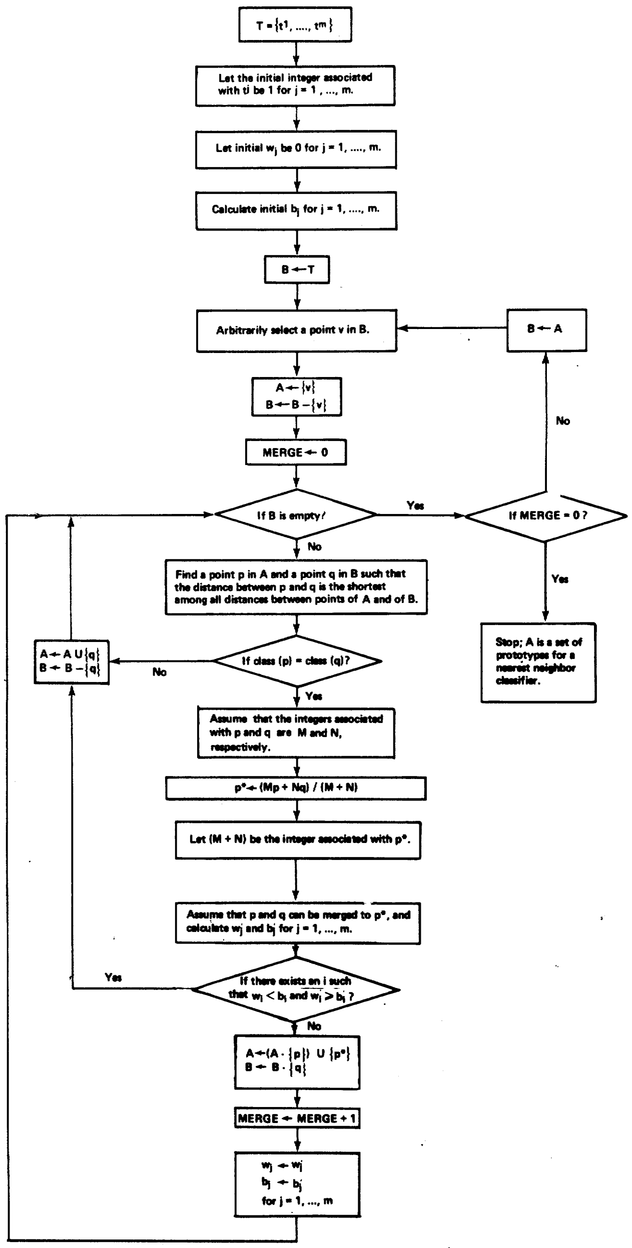

T = {t1, ..., tm}
Let the initial integer associated with tj be 1 for j = 1, ..., m.
Let initial wj be 0 for j = 1, ..., m.
Calculate initial bj for j = 1, ..., m.
B ← T
Arbitrarily select a point v in B.
B ← A
A ← {v}
B ← B − {v}
No
MERGE ← 0
If B is empty?
Yes
If MERGE = 0 ?
No
Find a point p in A and a point q in B such that the distance between p and q is the shortest among all distances between points of A and of B.
Yes
A ← A U {q}
B ← B − {q}
If class (p) = class (q)?
No
Stop; A is a set of prototypes for a nearest neighbor classifier.
Yes
Assume that the integers associated with p and q are M and N, respectively.
p* ← (Mp + Nq) / (M + N)
Let (M + N) be the integer associated with p*.
Assume that p and q can be merged to p*, and calculate wj and bj for j = 1, ..., m.
Yes
If there exists an i such that wi < bi and wi > bi ?
No
A ← (A − {p}) U {p*}
B ← B − {q}
MERGE ← MERGE + 1
wj ← wj
bj ← bj
for j = 1, ..., m

Fig. 3.

2(b)–(g) shows the sequence of $b_1,\cdots,b_6$ being updated. We see that the final $b_1,\cdots,b_6$ shown in Fig. 2(g) are the correct distances between $t^1,\cdots,t^6$ and their respective nearest points of different classes.

The detailed flowchart of the algorithm W^* is shown in Fig. 3.

III. EXPERIMENTS

The algorithm W^* given in the above section was implemented by a Fortran program. We give the following examples to show how well the program worked.

Example 1: Consider a training set T of 2-dimensional samples shown in Fig. 4(a). There are three classes. Each class has two clusters. There are all together 66 samples. We initially use all of these 66 samples as prototypes. However, after the program was applied to these prototypes, the number of prototypes was reduced to only 6. The 6 prototypes are shown in Fig. 4(b). The recognition rate for set T based on these 6 prototypes is the same as the one based on the initial 66 prototypes. Nevertheless, to classify a pattern, we now only have to compute 6 distances, instead of 66 distances initially needed. This is a saving of about 91 percent in computation.

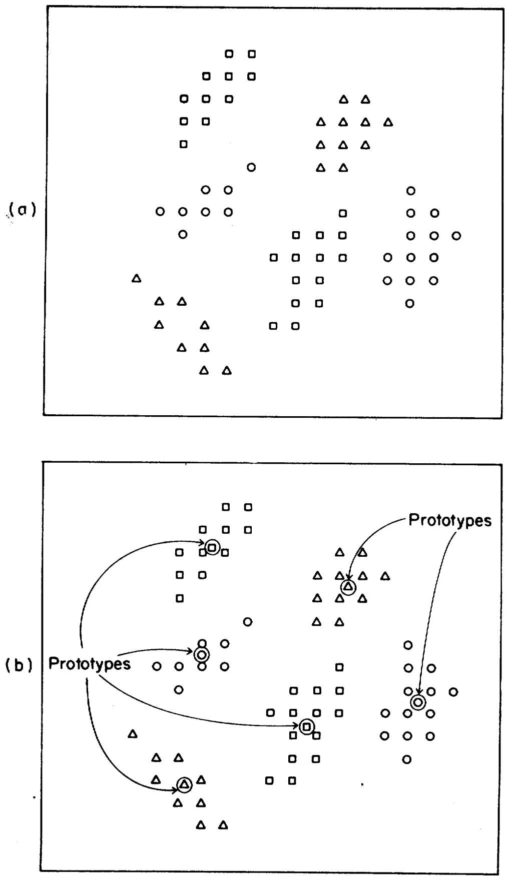


Fig. 4.

Example 2: In this example, we consider the iris data used by Fisher [4]. Four measurements, namely, sepal length, sepal width, petal length, and petal width, were made on an iris flower. There are three classes (varieties) of iris flowers, namely, Iris setosa, Iris versicolor, and Iris virginica. Fifty samples were obtained from each of the three classes. Thus, the training set consists of 150 samples. Our program started with 150 prototypes. However, after the program finished the job, only 14 prototypes were found to be needed to have the same recognition rate for the training set as the one using 150 prototypes. This is a reduction by about 90 percent.

Example 3: Consider the data set shown in Fig. 5(a). This set is similar to the one considered in [5]. We considered only 476 points because we were unable to generate 482 points as used by Hart. We ran our program on this set and reduced the number of prototypes to 17. The final prototypes and the decision boundaries are shown in Fig. 5(b). Our results are comparable with Hart's. In fact, in algorithm W^* shown in Fig. 3, if we arbitrarily

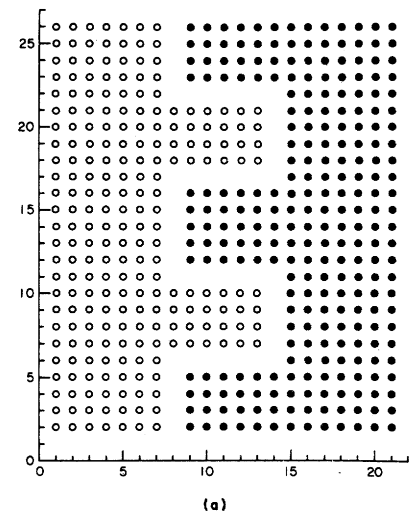
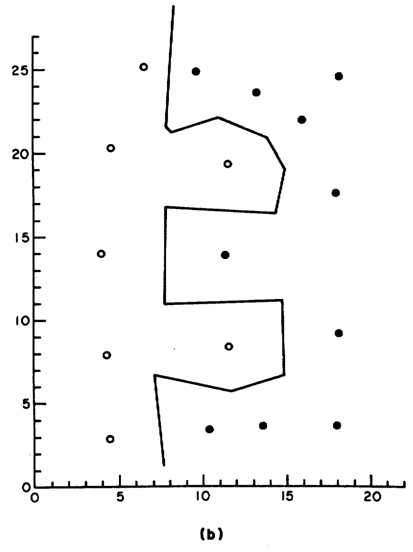



Fig. 5.

select a point p in A and a point q in B, and let p^* be p, instead of $(Mp + Nq)/(M + N)$, then algorithm W^* works similarly to Hart's algorithm except it is based on a different merging criterion. In algorithm W^*, we choose a pair of nearest neighbors p and q to merge because we believe that it is more likely to obtain a successful merging for p and q than for any random pair of points.

IV. THE DIAGNOSIS OF LIVER DISEASE

In this section, we shall consider a set of liver disease data. This set is a part of a large file used by Croft in his study [3]. Because of the limit of computer memory, we used only cases with 1 of the first 3 diseases out of 20 considered by Croft. Since there are missing data about some symptoms in Croft's file, we disregarded these symptoms and used a slightly different list of symptoms shown in Table I. After cases with missing information were eliminated from the training and test samples in Croft's file, the number of cases in each class for both our training and test sets is shown in Table II. Each case was checked for the presence or absence of the symptoms listed in Table I. We coded the presence by 1 and the absence by 0. Thus each case was characterized by a 45-dimensional binary vector. First, starting with all 514 cases of the training set as prototypes, we ran the program on a PDP-10 time sharing system. After the program was finished, 34 prototypes were produced. These 34 prototypes were then used to classify patterns in both the training and test sets. The recognition rates using the 514 initial prototypes were compared with the ones using the 34 final prototypes. These are given in Table III. From this table, we can see that a nearest neighbor classifier based on the 34 prototypes still gives high recognition rates for both the training and test sets, even though the number of the prototypes is only about 6 percent of the 514 initial prototypes.

TABLE I
LIST OF SYMPTOMS

Symptom Number	Symptom	Symptom Number	Symptom
1	Heavy alcoholic intake	24	Regeneration: paren. or mitoses
2	Nausea	25	Degeneration: diff. or focal
3	Weight loss	26	Degeneration: central or portal
4	Abdominal pain	27	Cells: diff. or focal
5	Malnutrition	28	Cells: Central or portal
6	Jaundice	29	Cells: polys
7	Ascites	30	Cells: lymphs
8	Edema	31	Cells: monos. or epithel
9	Abdominal collaterals	32	Cells: eos
10	Spider nevi	33	Cells: plasma or giant
11	Gynecomatis,	34	Fat: diff. or zonal
12	Testicular atrophy	35	Fat: 1-2$^+$ or 3-4$^+$
13	Hair loss	36	Pigment: Iron or bile
14	Palmar erythema	37	Pigment: paren. or gen.
15	Splenomegy	38	Pigment: kupff. or portal
16	Liver tenderness	39	Mall. B. or culture
17	Liver nodularity	40	Fibrosis: diff. or focal
18	Abnormal alkaline	41	Fibrosis: portal or cent.
19	Necrosis: diff. or focal	42	Stool color
20	Necrosis: Cent. or portal	43	Body temperature
21	Bile thrombi	44	1'Bilirubin
22	Regeneration: bile ducts	45	Total bilirubin
23	Regeneration: retic. endo.		

TABLE II
LIST OF DISEASES

Disease name	Disease number	Number in training set	Number in test set
Normal liver	1	103	26
Laennec's cirrhosis	2	345	80
Biliary cirrhosis	3	66	14
Total		514	120

V. CONCLUSION

We have given an algorithm for finding a small number of prototypes for a nearest neighbor classifier without sacrificing the recognition rate. The experimental results indicate that the algorithm is effective.

We note that if a classifier which decides membership of a pattern by a majority vote of the k nearest prototypes, we call it a k-nearest neighbor classifier [6]. A nearest neighbor classifier is just a 1-nearest neighbor one. It is easy to see that the algorithm given in this paper can be slightly modified to find prototypes for a k-nearest neighbor classifier.

Another point we would like to mention here is that a

TABLE III
RECOGNITION RATES FOR LIVER DISEASE DATA

Classifiers	Training Set (514 cases) Recognition Rate (%)	Training Set (514 cases) Error Rate (%)	Test Set (120 cases) Recognition Rate (%)	Test Set (120 cases) Error Rate (%)
The Nearest Neighbor Classifier Using 514 Initial Prototypes	100	0	92.5	7.5
The Nearest Neighbor Classifier Using 34 Final Prototypes	100	0	91.7	8.3

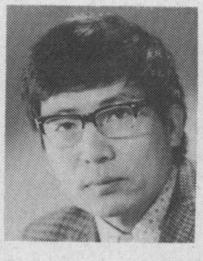
nearest neighbor classifier can be changed into a piecewise linear classifier [1], [6]. That is, in the nearest neighbor discriminant function g_i given in (1) of Section I, if we replace every $d(x,p_i^j)$ by $(-x\cdot p_i^j + 0.5p_i^j\cdot p_i^j)$, we will obtain a piecewise linear discriminant function, denoted by g_i^*. It can be shown that the classification decision of a pattern based on $g_1^*,\cdots,g_r^*$ is the same as the one based on $g_1,\cdots,g_r$. Therefore, we can use the algorithm given here to find piecewise linear discriminant functions. That is, first, use the algorithm to find nearest neighbor discriminant functions, and then change them into piecewise linear ones. In this way, the number of linear functions in the piecewise linear discriminant functions need not be specified beforehand.

ACKNOWLEDGMENT

The author wishes to thank Dr. Croft of the University of Utah for the use of his liver disease data file. He also wishes to thank Dr. Slagle of National Institutes of Health for useful comments and discussion.

REFERENCES

[1] C. L. Chang, "Pattern recognition by piecewise linear discriminant functions," *IEEE Trans. Comput.*, vol. C-22, pp. 859–862, Sept. 1973.
[2] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 21–27, Jan. 1967.
[3] D. J. Croft, "Is computerized diagnosis possible?" *Comput. Biomed. Res.*, vol. 5, pp. 351–367, 1972.
[4] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugen.*, vol. 7, pp. 178–188, 1936.
[5] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory* (Corresp.), pp. 515–516, May 1968.
[6] W. S. Meisel, *Computer-Oriented Approaches to Pattern Recognition.* New York: Academic, 1972.
[7] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, pp. 1389–1401, Nov. 1957.
[8] G. J. S. Ross, "Minimum spanning tree," *Appl. Statist.*, vol. 18, no. 1, pp. 103–104, 1969.
[9] C. T. Zahn, "Graph-theoretical methods for detecting and describing Gestalt clusters," *IEEE Trans. Comput.*, vol. C-20, pp. 68–86, Jan. 1971.



Chin-Liang Chang (S'66–M'67) was born in Taiwan, China, on May 28, 1937. He received the Diploma from the Taipei Institute of Technology, Taiwan, China, in 1958, the M.S. degree from Lehigh University, Bethlehem, Pa., in 1964, and the Ph.D. degree from the University of California, Berkeley, in 1967, all in electrical engineering.

From 1964 to 1967 he was a Teaching and Research Assistant in the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. From 1967 to 1974 he was with the Heuristics Laboratory, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Md. Since July 1974 he has been with IBM Research Laboratory, San Jose, Calif. He is a coauthor of a book entitled Symbolic Logic and Mechanical Theorem Proving and has written several articles. His research interests are information systems, theorem proving, pattern recognition, and artificial intelligence.

Dr. Chang is a member of the Association for Computing Machinery.