A Search Technique for Pattern Recognition Using Relative Distances

Thomas E. Portegys

Abstract—**A technique for creating and searching a tree of patterns using relative distances is presented. The search is conducted to find patterns which are nearest neighbors of a given test pattern. The structure of the tree is such that the search time is proportional to the distance between the test pattern and its nearest neighbor, which suggests the anomalous possibility that a larger tree, which can be expected on average to contain closer neighbors, can be searched faster than a smaller tree. The technique has been used to recognize OCR digit samples derived from NIST data at an accuracy rate of 97% using a tree of 7,000 patterns.**

Index Terms—**Pattern recognition, optical character recognition, nearest neighbor, distance metric, branch and bound, NIST digit samples.**

I. Introduction

A technique for creating and searching a tree of patterns using relative distances is presented. The search is conducted to find patterns which are nearest neighbors of a given test pattern. The structure of the tree is such that the search time is proportional to the distance between the test pattern and its nearest neighbor, which suggests the anomalous possibility that a larger tree, which can be expected on average to contain closer neighbors, can be searched faster than a smaller tree. The technique has been used to recognize Optical Character Recognition (OCR) digit samples derived from National Institute of Standards and Technologies (NIST) data [11] at an accuracy rate of 97% using a tree of 7,000 patterns.

The task of recognizing handwritten characters is an active area of research, especially in neural networks [4], [8], [10]. This paper is an investigation of a how a particular memory-based, nearest neighbor search technique behaves when applied to a large number of patterns. As a nearest neighbor scheme, the search technique attempts to address problems encountered in dealing with many-dimensional objects [3], [5], [9], which in this case are represented by OCR patterns. The technique is intended to be applicable not only to character recognition, but to pattern recognition tasks in general.

The paper is organized as follows: First, a description of the technique is presented, including the definition of distance, the insertion algorithm, and the search algorithm. The results of the NIST and other tests are then presented. Finally, a proposal is made for a device to improve the speed of searching.

II. Description

A. Distance Formula

The distance between patterns P1 and P2 was chosen to be the city-block distance:

$$dist(P1, P2) = \sum_{i=1}^{N} |p1_i - p2_i| \quad (1)$$

where

N = number of pixel in the patterns
p1, p2 = pixel values

The city-block distance, which is the Hamming distance for binary pixel values, is possibly not the best choice; it was chosen for the value of its fast computation. Any distance formula is suitable if it conforms to these conditions:

$$dist(P1, P2) \geq 0 \quad (i)$$
$$dist(P1, P2) = dist(P2, P1) \quad (ii)$$
$$dist(P1, P3) \leq dist(P1, P2) + dist(P2, P3) \quad (iii)$$

Conditions (i) and (ii) hold for the city-block distance due to the absolute value operator. Condition (iii) is the well-known triangle inequality [6]. This condition must hold for patterns containing single pixels since it must be true for the distances between any three scalar numbers. It must then hold for multiple pixel patterns since inequality relationships are preserved when summing.

B. Pattern Insertion

Patterns are stored in a tree structure according to their relative distances for efficient searching. They are inserted into the tree by a recursive procedure starting at the root which is the first prospective parent pattern. A decision is made whether to link the new pattern directly to the parent pattern or to pass it on to the first child of the parent to which it "fits." An inserted pattern fits a child pattern if the distance between it and the child is less than the distance between the parent and the child multiplied by a constant. If the constant is .5, for example, the node is passed to the child if it is within a radius of half the distance between the parent and the child patterns. Once passed to a child, the child becomes the parent for the next iteration of link checking.

The algorithm used for insertion, written in C, using RADIUS as the link control constant, is given in Appendix A.

The purpose of the RADIUS constant is to control the degree (brushiness) of the tree. At the extremes, when RADIUS is set to 0, all children are linked to the root pattern; when set to 2, no pattern will have more than one child, i.e., the tree will be a linked list. For the NIST tests described later, RADIUS was set to .7, which an average node degree of 3.6.

One feature of the algorithm is that the order of a pattern's subtree branches is important: A new pattern is always inserted in the first fitting branch (even though there can be more than one such branch). This features allows the search of a tree which contains a duplicate of a given pattern to proceed with maximum efficiency: The duplicate will always be found on the first branch which fits the pattern.

Another feature is a tree reorganization procedure which prevents excessive children from accumulating on a parent pattern. When a new child is linked to a parent pattern, every other previously inserted child is checked to determine if it should be linked to the new child instead of the parent. For each such child, the child subtree is severed from the parent and each pattern in the subtree is inserted at the current parent pattern. Note that they are not inserted into the new child since not all patterns in the subtree necessarily fit there.

C. Pattern Searching

The purpose of the pattern search algorithm is to efficiently find patterns in the tree which are nearest neighbors of a given test pattern. Searching is done in a best-first manner, that is, the pattern whose subtree could contain the pattern least distant from the search pattern is searched next.

The essence of the search decision procedure is illustrated in Fig. 1. When the search pattern S arrives at pattern A, it must determine whether to search either pattern B or D next. It does this by calculating the least distances between S and the patterns which can potentially appear within the subtrees of B and D. These are labeled B′ and D′, respectively. This distance, called the search distance, is defined as follows for S and B:

Manuscript received Dec. 20, 1993; revised Mar. 17, 1995.
The author is with AT&T Bell Laboratories, Naperville, IL 60566; e-mail: t.e.portegys@att.com.
IEEECS Log Number P95091.

$$search(S, B) = dist(S, B) - (dist(A, B) * RADIUS) = dist(S, B') \quad (2)$$

RADIUS is the link control constant (see the pattern insertion algorithm). If (2) results in a value less than zero, the search distance is set to zero. After calculating the search distances for B and D, the search proceeds to the pattern having the least search distance.

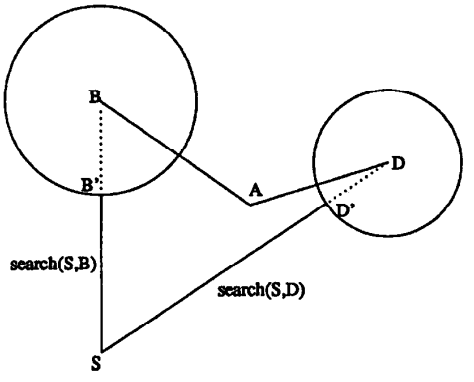

Fig. 1. Search distance.

The search algorithm is given in Appendix B. The SEARCH-WORK structures contain temporary data and are configured into a copy of the searched portion of the pattern tree during the search. To prevent searching the entire tree, a maximum parameter can be provided. This value of this parameter can be changed dynamically to allow for more or less searching under various circumstances, such as time constraints. A list of similar patterns is returned by the algorithm.

The algorithm features a branch and bound capability which allows portions of the search tree to be cut-off during the search: As less distant patterns are found, greater cut-off is achieved. This is because a pattern does not have to be searched if its search distance is greater than the distance of a previously found pattern.

III. Digit Recognition Test Results

The patterns were derived from NIST digits 0-9, and were obtained through an internal company source. They were size normalized to fit in a 20 × 20 pixel box, and were then centered to fit in a 28 × 28 image using center of gravity. The pixels were scaled to four levels of gray value. Fig. 2 shows an example of a pattern for the digit '6.' The comparisons were done without any translation or rotation.

A. Test 1

For the first test, approximately 7,000 digit samples were inserted into a tree, and a different set of 1,000 patterns was selected as test patterns. 97.1% of the test samples were identified correctly, which is comparable with other pattern recognizers. In addition, on the average, the nearest neighbor was found after searching 337 patterns in the search tree, and 6,000 patterns were not searched due to cut-off conditions.

B. Test 2

The next series of tests were an attempt to simulate a search tree large enough to presumably contain patterns that are very similar to a set of search patterns. The question of what happens to the reliability and extent of the search of such trees in relation to their size was the focus of these tests.

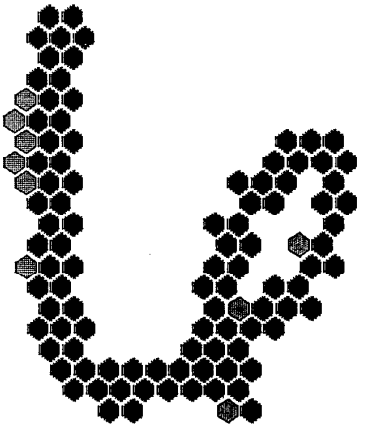

Fig. 2. Example NIST digit.

The initial search tree contained 1,000 patterns, and these same 1,000 patterns also comprised the search set, meaning the search was to find identical patterns in the tree. Following this, an additional 1,000 patterns were inserted into the tree, and a random set of 1,000 out of the 2,000 accumulated patterns was selected as the search set. This procedure was repeated until 10,000 patterns were inserted into the tree.

Fig. 3 shows the average number of patterns searched before finding the identical pattern as a function of search tree size. In all cases, the identical pattern was found. It can be seen that only about 25 additional patterns were searched as the tree grew by 9,000 patterns. The data also roughly conforms to the function $\log_{3.6}(x)*10$, where 3.6 was found to be the average degree of the search tree. This suggests that the search effort is proportional to some logarithmic function of the tree size.

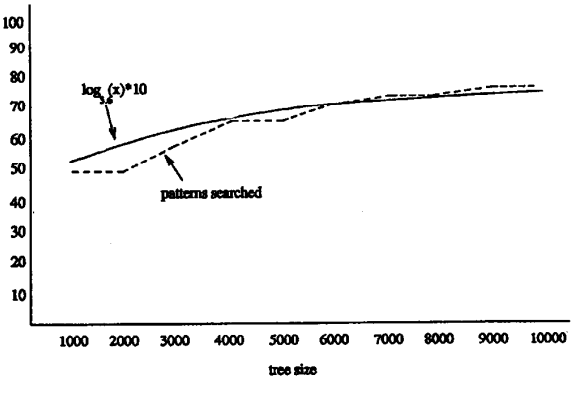

Fig. 3. Patterns searched to find identical pattern.

C. Test 3

As a check on the validity of using identical search patterns instead of similar ones, noise was randomly introduced into the search patterns such that they were closely similar, but not identical to, patterns in the tree. The noise was chosen such that the average distance between a stored pattern and a modified search pattern was 10% of the average distance between the stored pattern and a random pattern. Searching a tree containing 10,000 patterns resulted in a 100% identification rate, and an average search of 196 patterns to find the most similar. In addition, 9,065 patterns were not searched due to cut-off conditions.

A forced cut-off at 200 patterns was introduced in the above search to determine the effect of limiting the extent of the search. This resulted in an identification rate of 96%, with the most similar pattern being found after an average search of 79 patterns. 9,801 patterns were not searched due to cut-off conditions. The primary reason for the effectiveness of the limit was that it curtailed the relatively few searches of excessive extent. In most cases, these excessive searches were not necessary since a correctly identifying pattern was found early on, even though it was not the most similar one.

D. Test 4

To look further at the effect of increasing the distance between the search patterns and the stored patterns in a larger tree, 2^{17}(131,072) patterns were created by generating all possible combinations of the pixel values of 0 and 10, thus ensuring that the stored patterns have a minimum distance of 10 between them. These were then added to the search tree in random order. A set of search patterns was created by randomly selecting 100 stored patterns. This formed the distance 0 search set. The distance 1 search set was formed by copying the distance 0 set, randomly selecting a pixel in each pattern and modifying its value by 1. Distance 2-5 search sets were created in successive manner.

Fig. 4 plots the number of patterns searched to find the most similar stored pattern as a function of the distance of the search patterns. The function appears to be a linear one, especially looking at the distance 2-5 points.

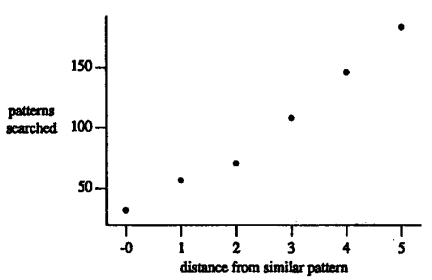

Fig. 4. Patterns searched to find similar pattern in 131072 stored patterns.

IV. A Pattern Comparator Device

The search mechanism demands the repetitive computation of (1), which for the 784 pixel NIST pattern size requires a little over 5 ms per invocation on a SUN SPARC workstation. This is the time to compare two patterns during a search which involves multiple comparisons. In approximately the same time other systems are able to complete a recognition computation using specialized hardware [12]. It seems clear that there is a need for a faster means of computing the distance. A vector processor would be able to compute the difference terms in parallel, but the summation of these terms would remain as a bottleneck.

An optical device may hold the answer as a means of performing the summation. Consider the device shown in Fig. 5. The difference terms are transduced into optical signals whose intensities are proportional to the sizes of the differences. A lens is used to focus these signals onto a detector which is capable of responding in proportion to the sum of the signal intensities, thus achieving a naturally parallel summation of the terms.

It is likely that this device could be made to compute a sum in a few microseconds, given the known performance of optical devices [1], [7].

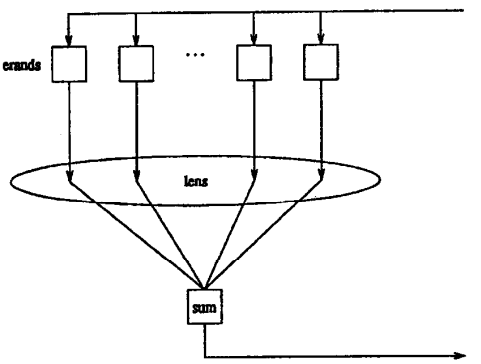

Fig. 5. Optical summation.

V. Conclusion

Much like chess playing machines, which have gained high rankings largely by relying on speed [2], the findings presented here suggest that a "brute force" approach, in the form of storing a large number of patterns, may be effective for pattern recognition.