SPTA: A Proposed Algorithm for Thinning Binary Patterns

NABIL JEAN NACCACHE AND RAJJAN SHINGHAL, MEMBER, IEEE

Abstract—**A new skeletonization algorithm called the safe-point thinning algorithm (SPTA) is proposed. It is suitable for binary patterns. The SPTA is informally explained and is also defined in a formal algorithmic manner. The experimental performance of the SPTA is compared with fourteen other skeletonization algorithms that have been proposed by other researchers. Results show that SPTA is the fastest and produces skeletons of good quality. Moreover, any skeleton produced has enough information such that a pattern can be reconstructed from it, if so desired. The reconstructed pattern has been found to be nearly identical to the original pattern from which the skeleton was derived.**

I. INTRODUCTION

IN PROCESSING binary patterns, skeletonization consists of iterative deletions of the dark points (i.e., changing them to white) along the edges of a pattern until the pattern is thinned to a line drawing. The thinned pattern (called a skeleton) must preserve the connectedness and shape of the original pattern [11]. It should be noted that the skeleton of a pattern may not be unique [4]. Ideally the original pattern should be thinned to its medial axis [4], [11]. One advantage of skeletonization is to reduce the memory space required for storing the essential structural information present in the patterns. Secondly, it also simplifies the data structures required in processing the patterns [4]. During the past few years, many researchers have proposed different skeletonization algorithms. These algorithms have been shown to be of great interest in optical character recognition [3]–[5], [7], [11], [15], chromosome analysis [8], and fingerprint classification [9].

In Section II we will briefly review 14 known skeletonization algorithms as described by eight other researchers or groups of researchers. In Section III we describe a new skeletonization technique (called the safe-point thinning algorithm) suitable for binary patterns. To improve the speed of our algorithm, we adopted some special techniques in its implementation. These techniques are described in Section IV. Experimental results comparing the performance of our algorithm with the other known skeletonization algorithms are given in Section V. Lastly in Section VI we conclude by discussing the efficacy of our skeletonization algorithm.

Manuscript received July 11, 1983; revised November 1983. This work was supported by the National Science and Engineering Research Council of Canada under Grant A3060.

The authors are with the Department of Computer Science, Concordia University, 1455 De Maisonneuve Blvd., West Montreal, Quebec, Canada, H3G1M8.

II. HISTORICAL REVIEW OF SKELETONIZATION ALGORITHMS

Since we are reviewing as many as 14 algorithms, and since we do not want this paper to get unnecessarily long, our review may not be able to explain these historical algorithms in every single detail. We give only the essence of each of these algorithms. Our comments on the empirical performance of these algorithms are given in Section V.

Each element in a binary pattern can be either a *dark point* or a *white point*. Skeletonization usually consists of iteratively deleting edge-points (dark points along the edges of a pattern), such that the deletion of these points 1) does not remove end-points (intuitively, end-points are dark points at the open extremeties of a stroke); 2) does not break the connectedness of the pattern; and 3) does not cause excessive erosion (e.g., a stroke is not iteratively deleted). The researchers of [1]–[3], [8], [11], [13], [15], [16] usually differ in the manner in which they conduct the tests to meet these criteria. There may be a partial commonality in the approaches of these researchers, but no two of their algorithms can be considered identical. Each researcher's success in skeletonization thus depends on the goodness of the tests conducted. But first, we establish some notation that will facilitate our review.

The eight-neighbors of point p are defined to be the eight points adjacent to p (points n_0 to n_7 in Fig. 1). Points n_0, n_2, n_4 and n_6 are also referred to as the four-neighbors of p. Some researchers prefer to call the four-neighbors as the orthogonal neighbors [3]. A skeleton is said to be j-connected (j is equal to four or eight) if between any two dark points p_0 and p_n there exists a path $p_0p_1 \cdots p_{i-1}p_i \cdots p_n$ such that p_{i-1} is a j-neighbor of p_i for $1 \leqslant i \leqslant n$. All the algorithms that we are reviewing below define formally an *end-point* as a dark point with at most one dark eight-neighbor.

Arcelli's Algorithm [1] (ARCL): A dark point p is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; and 3) its neighborhood satisfies two predefined boolean expressions, where each boolean variable represents the state (darkness or whiteness) of a neighbor of p. This is called the break-point test; it is done to ensure the connectedness of the skeleton.

Bel-Lan's and Montoto's Algorithm [2] (BELA): A dark point p is deleted from the pattern if it satisfies all of the

0018-9472/84/0500-0409$01.00

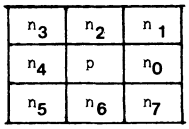

Fig. 1. A point p and its neighborhood.

following conditions: 1) it has at least one white four-neighbor; and 2) the neighborhood of p does not match any of eight predefined 3×3 windows. This conducts the break-point test and also ensures that the point to be deleted is not an end-point (this is called the end-point test).

Beun's Algorithm Using Orthogonal Neighbors [3] (BEUO): A dark point p is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; and 3) its neighborhood does not match any of seven predefined 3×3 windows. This conducts the break-point test and also prevents excessive erosion, called the excessive erosion test.

Beun's Algorithm Using Saraga-Woollons' Criteria [3] (BEUS): A dark point p is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e., its neighborhood matches any of four predefined 3×3 windows, called the Saraga–Woollons criteria [14]; 2) its deletion does not cause the introduction of a loop; i.e., a white point surrounded by dark points; and 3) its neighborhood does not match another set of seven predefined 3×3 windows (break-point and excessive erosion tests).

Hilditch's Algorithm [8] (HILD): A dark point p is deleted if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; 3) it has at least one dark eight-neighbor not considered as deletable; 4) it is not a break-point; i.e., its neighborhood satisfies a predefined "crossing-number" criterion; and 5) its deletion does not cause excessive erosion.

Pavlidis' Classical Algorithm [11] (PAVC): A dark point p is deleted from the pattern if it satisfies all of the following criteria: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; and 3) its neighborhood does not match any of two predefined 3×3 rotating windows (break-point test).

Pavlidis' Basic Algorithm [11] (PAVB): A dark point p is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; and 3) its neighborhood does not match any of three predefined 3×3 rotating windows (Pavlidis calls these "multiple point" and "tentative multiple point" tests; effectively, these tests ensure connectedness of the skeleton and prevent excessive erosion).

Pavlidis' Reconstructability Algorithm [11] (PAVR): A dark point p is deleted from the pattern if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white four-neighbor; 2) it is not an end-point; and 3) its neighborhood does not match any of four predefined 3×3 rotating windows, three of which are identical to the windows used in PAVB. It can be seen that PAVR is similar to PAVB except in the following additional details: a) the presence of an extra window in PAVR causes a difference in the number of points deleted in both algorithms; b) PAVB does an extra scan over the pattern to delete some of the points that satisfy Pavlidis' "tentative multiple point" test; this extra scan is not done in PAVR; 3) in PAVR, the points that are retained are so labeled that the skeleton can suitably be used to reconstruct the original pattern. Pavlidis has described his reconstruction algorithm that we will examine in greater detail in Sections IV and V.

Stefanelli–Rosenfeld's Version of Hilditch's Algorithm [15] (SRVH): Stefanelli and Rosenfeld presented what they called a "simplified version" of the Hilditch algorithm described above. In this version, a dark point p is deleted if it satisfies all of the following conditions: 1) it is an edge-point; i.e., it has at least one white eight-neighbor; 2) it is not an end-point; 3) its deletion does not cause excessive erosion; and 4) it satisfies the break-point test; i.e., the number of dark to white transitions in the neighborhood of p is equal to 1.

Stefanelli's and Rosenfeld's Algorithm with Four Scans per Pass [13], [15] (SR4S): In the first scan across the pattern, bottom edge-points (i.e., they have n_6 as white) are deleted if their neighborhoods do not match any of six predefined 3×3 windows. Subsequent scans similarly delete top, left and right edge-points. The scans are iteratively repeated till the pattern is thinned to a skeleton.

Stefanelli's and Rosenfeld's Algorithm with Two Scans per Pass [13], [15] (SR2S): This algorithm is somewhat similar to SR2S. In the first scan, a dark point p is deleted if it is either a bottom or a left edge-point and if its neighborhood does not match any of nine predefined 3×3 windows. The second scan similarly deletes top and left edge-points.

Tamura's Algorithm, Ensuring Four-Connectedness [16] (TM4F): Tamura proposed conducting the same four scans as in SR4S but modifying the windows slightly. This modification he said would eliminate the "mistakes" of SR4S. Tamura does not explain clearly what "mistakes" he found in SR4S. He cites two other papers of his that we have not been able to read as the papers are in Japanese, a language we do not know.

Tamura's Algorithm, Ensuring Eight-Connectedness [16] (TM4E): Tamura then further modified his own algorithm TM4F by using another set of windows but retaining the four scans across the pattern. By changing the windows, Tamura showed that whereas TM4F gave four-connected skeletons, TM4E gave eight-connected skeletons.

Tamura's Algorithm, with Two Scans per Pass [16] (TM2E): Just as he did for SR4S, Tamura suggested changes to SR2S also. He thus proposed conducting the same two scans as in SR2S but modifying the windows slightly. The skeletons produced were eight-connected just as they were so by TM4E.

We concede that there exist other skeletonization algorithms which we have not reviewed above. However, most

of those algorithms have already been investigated and reported in [4], [16]. We realized that those algorithms will not be suitable for our investigation and so we did not review them above.

In the following section we give a detailed description of the skeletonization algorithm that we are proposing. Later in Section V, empirical performance of our algorithm will then be compared with the 14 algorithms reviewed above.

III. The Proposed Algorithm

Our objective is to develop a skeletonization algorithm that will be fast, that will prevent excessive erosion, and one that will give good skeletons (i.e., skeletons that retain the shape information of the original pattern and do not have spurious tails) [4]. Furthermore, the algorithm should be such that enough information is retained in the skeleton to reconstruct as far as possible a pattern that is similar to the original pattern. This is called *reconstructability*. The usefulness of this ability in image processing has been discussed in [4].

Although we have intuitively explained edge-points, end-points, and break-points above in Section II, we now give their definition as it pertains to our proposed algorithm (called the *Safe-Point Thinning Algorithm*, or SPTA for short): 1) an *edge-point* is a dark point that has at least one white four-neighbor; 2) an *end-point* is a dark point that has at most one dark eight-neighbor; and 3) a *break-point* is a dark point, the deletion of which would break the connectedness of the original pattern.

It is assumed that the input pattern is a smoothed pattern; that is, all pattern irregularities and all salt-and-pepper noise have been removed [6], [17]. In essence, the SPTA consists of executing many passes over the pattern, where in each pass a few dark points are *flagged*. A *flagged point* must be such that it is an edge-point, but not an end-point, nor a break-point, and nor must its possible deletion cause excessive erosion in the pattern. At the end of the pass, all flagged points are deleted. If no points are deleted at the end of a pass, then the skeletonization procedure stops. The SPTA differs from the algorithms reviewed in Section II in the manner in which the tests are conducted and the details of implementation, all of which are described below. For the sake of easy readability, initially we shall present our description in a semi-formal manner; the formal description appears later in Appendix I.

The SPTA identifies an edge-point as one or more of the following four types: 1) a left edge-point, having its left neighbor n_4 white (see Fig. 1); 2) a right edge-point, having n_0 white; 3) a top edge-point, having n_2 white; and 4) a bottom edge-point, having n_6 white. It should be noted that an edge-point can be more than one type; for example, a dark point having n_2 white and n_4 white will be a top edge-point and a left edge-point simultaneously.

In the following discussion, we will refer exclusively to left edge-points. The discussion is later generalized to other types of edge-points.

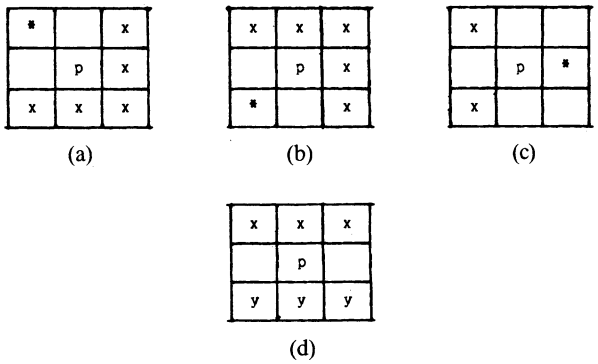

Fig. 2. If the neighborhood of a dark point p matches any of the four windows above, then SPTA does not flag p. '*' indicates a dark point. x's and y's are "don't cares" (their whiteness or darkness is immaterial).

To flag a left edge-point p, the SPTA must first show that p is not an end-point, nor a break-point, and nor would its deletion cause excessive erosion. We show below that to identify such a left edge-point p, SPTA needs to compare the neighborhood of p with the four windows shown in Fig. 2. If the neighborhood of p matches any one of the four windows, then p is not flagged. It should be noted that the points shown as x's and y's in the windows are "don't care" points (i.e., their whiteness or darkness is immaterial). We now examine the four windows one by one, to justify why these are the windows required by the SPTA to conduct the break-point, end-point and excessive erosion tests on p.

If the neighborhood of p matches any of the windows (a), (b) or (c) of Fig. 2, then two situations may occur: 1) if all x's are white, then p is an end-point; 2) if at least one of the x's is a dark point, then p is a break-point. Thus, in either of these cases, p should not be flagged.

Now let us examine window (d) of Fig. 2. If at least one x and at least one y are dark, then p becomes a break-point and thus it should not be flagged. For further analysis, let us assume for the time being that all x's are white. Then there are eight possible configurations as shown in Fig. 3. Configurations w_1, w_2, and w_3 make p an end-point, and configuration w_4 makes p a break-point. The possible deletion of p in configurations w_5 and w_6 could cause excessive erosion in slanting strokes of width 2; e.g., Fig. 4(a) being reduced to Fig. 4(b). In configurations w_7 and w_8, the point p is in fact noise. Since the patterns have already been smoothed [6], [17] before they are fed to our SPTA, such noise is always removed and configurations w_7 and w_8 shall never exist at the beginning of the skeletonization process. If configuration w_7 were to occur in an intermediate stage of skeletonization, then p would be a spur due to a short tail in the original pattern (e.g., as in chromosomes). Such a point may, however, retain some shape information of the pattern and it must not be deleted. Similarly, if configuration w_8 were to occur in an intermediate stage of skeletonization, then it would be a singleton (isolated point); its deletion would totally erase the last remaining segment of the pattern. Therefore, in all of the above configurations p must not be flagged. By

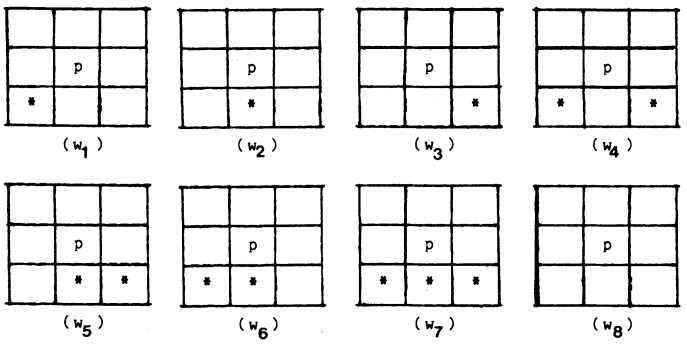

Fig. 3. The eight possible configurations that could exist in the window (d) of Fig. 2, when all x's are white.

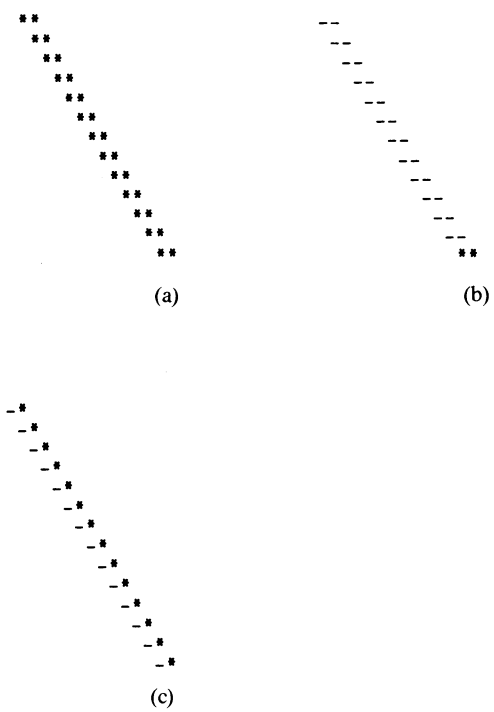

Fig. 4. A slanting stroke and the skeletons that could be obtained from it. '–' represents the deleted points. (a) A slanting stroke of width 2. (b) This is the skeleton obtained if we do either the following: if we flag a dark point p whose neighborhood matches configurations w_5 or w_6 of Fig. 3; or, if we perform only one scan per pass in the SPTA, flagging all four types of edge-points that do not satisfy the boolean expressions S_0 to S_6. Either approach gives excessive erosion. (c) The skeleton obtained by SPTA with two scans per pass.

symmetry, we extend our argument to the case in window (d) when all the y's are white and x's take on varying values of whiteness or darkness. Thus for window (d) also, x's and y's become "don't care" points.

Reinforcing our analytic arguments above, we have exhaustively verified by experiment that the four windows of Fig. 2 are the *only ones* we require to conduct end-point, break-point and excessive erosion tests on a dark point p.

A left edge-point which matches any one of the four windows of Fig. 2 is called a *left safe-point*. It can be easily shown that for a left safe-point the boolean expression S_4 is FALSE, where

$$S_4 = n_0 \cdot (n_1 + n_2 + n_6 + n_7) \cdot (n_2 + \bar{n}_3) \cdot (n_6 + \bar{n}_5).$$

A boolean variable has the value TRUE when its corresponding point is dark and unflagged, and it has the value FALSE otherwise (i.e., if the point is flagged or it is white). Thus, to test whether a left edge-point is a left safe-point, the SPTA needs only to test against boolean expression S_4. The subscript in S_4 indicates that the boolean expression was derived for left edge-points, i.e., they have their left neighbor n_4 white.

Similarly, we can derive the following Boolean expressions.

For Right Safe-Point:

$$S_0 = n_4 \cdot (n_5 + n_6 + n_2 + n_3) \cdot (n_6 + \bar{n}_7) \cdot (n_2 + \bar{n}_1);$$

For Top Safe-Point:

$$S_2 = n_6 \cdot (n_7 + n_0 + n_4 + n_5) \cdot (n_0 + \bar{n}_1) \cdot (n_4 + \bar{n}_3);$$

and,

For Bottom Safe-Point:

$$S_6 = n_2 \cdot (n_3 + n_4 + n_0 + n_1) \cdot (n_4 + \bar{n}_5) \cdot (n_0 + \bar{n}_7).$$

A pass in the SPTA consists of two scans, where a scan examines every point in the pattern. The scanning sequence may be either row-wise or columnwise, at the user's choice. In the first scan, all left edge-points and all right edge-points that are not respectively left safe-points and right safe-points are flagged. Similarly in the second scan, the corresponding top edge-points and bottom edge-points are flagged. If there were no flagged points the procedure stops, else it begins the next pass. A question arises at this point: would it not be better to have a single scan per pass, where the scan could flag all four types of edge-points that do not satisfy our boolean expressions S_0, S_2, S_4 and S_6? A detailed analytic argument, exhaustively examining different pattern configurations can be given as to why the single scan approach could give excessive erosion. Rather than go into a long, detailed, cumbersome explanation, we show an actual example which we implemented. The slanting stroke of Fig. 4(a) was skeletonized by the one-scan approach to a set of two dark points as shown in Fig. 4(b). That was clearly a case of excessive erosion. However, using our two-scan approach, the skeleton of Fig. 4(a) is shown in Fig. 4(c). Thus it is essential to have two scans per pass in SPTA.

To remove any ambiguity in our explanation, a Pascal-like algorithmic version of SPTA (assuming a row-wise scan) is given in Appendix I. It is trivial to make modification in that algorithmic representation to accomodate a columnwise scan. It should be noted that a skeleton produced by a row-wise scan may have minor variations as compared to a skeleton produced by a columnwise scan, given the same input pattern.

IV. Implementation of SPTA

Although our SPTA is conceptually clear in the description given in Section III above, there are details that need to be explained in actual implementation so that the algorithm could be speeded up as far as possible. We describe below three implementation techniques that we applied to achieve these objectives: 1) how the number of scans could

be minimized; 2) how a suitable decision tree could be developed to test the boolean expressions S_0 through S_6; and 3) how the points in the pattern were labeled during scanning. Our labeling technique not only speeded up our SPTA but also gave reconstructability. We discuss these three techniques one by one in detail below.

A. Minimizing Number of Scans

As described in our algorithm above, Scan 1 flags some left and right edge-points, and Scan 2 flags some top and bottom edge-points. Say Scan 1 flags no point but Scan 2 flags a few. The algorithm does not terminate yet, but in the following pass we do not initiate Scan 1. Only Scan 2 is initiated. This is because if Scan 1 did not flag any point in a given pass, it will not flag any point in a subsequent pass. Within any scan, we added a further refinement. Say in a given pass Scan 1 flagged only some left edge-points, but it flagged none of the right edge-points. Then, in the following pass, Scan 1 needs to test only the left edge-points for flagging. Similar refinements in Scan 2 helped in speeding up the skeletonization process of the SPTA.

B. Testing Boolean Expressions S_0 through S_6

As discussed in Section III above, the boolean expressions S_0 through S_6 are tested to decide on the safe-point status of a dark point p. It is apparent by looking at these boolean expressions that, to reach a decision, the number of points examined in the neighborhood of p depends upon the sequence in which these points are examined. Such a sequence may be heuristically decided by a user. For the data base of patterns described below in Section V, we found the decision trees given in Fig. 5 to be optimal. This means that the fewest number of points in the neighborhood were examined before reaching a decision. For our data base, we found this number to be on an average, 2.55.

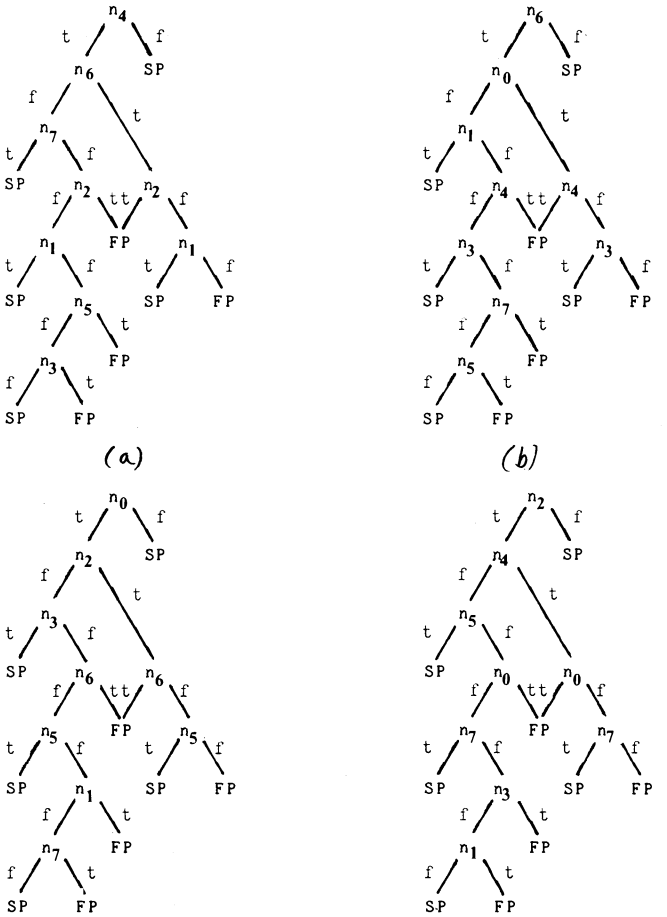

Fig. 5. The tree structure representing the boolean expressions for edge-points. n_i indicates the eight-neighbor being visited (where $0 \leqslant i \leqslant 7$); SP means that p becomes a safe-point; FP means that p becomes a flagged point. t stands for boolean value TRUE, and f stands for boolean value FALSE. (a) Decision tree for S_0. (b) Decision tree for S_2. (c) Decision tree for S_4. (d) Decision tree for S_6.

C. Labeling of Pattern Points and Reconstruction

Initially in the pattern all dark points are labeled zero and all white points are labeled minus MAXINT (where MAXINT is the largest possible integer storable in the computer). During any pass of the skeletonization, the value of the non-edge-points does not change. For the edge-points two situations occur: 1) if it is flagged it takes on the value of (i − MAXINT), where i is the iteration number of the pass in which the point was flagged, the first pass having a value $i = 1$ by convention; 2) if it is identified as a safe-point, then it is labeled by the value of i; that is, the label of a safe-point indicates the smallest iteration number of the pass in which the point was identified as a safe-point. For ease of understanding, the tree in Fig. 6 shows our labeling technique. With this labeling technique, we achieved two objectives that are described in the following.

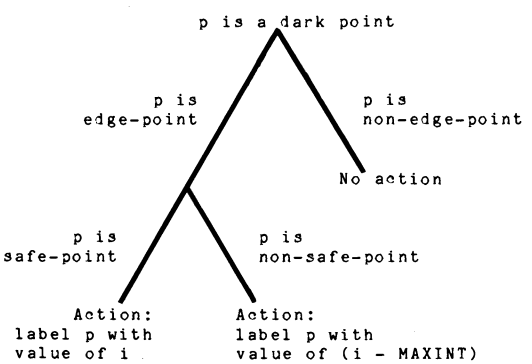

Fig. 6. The decision tree required to label a dark point. The value of i represents the iteration number of the skeletonization process over a pattern. MAXINT is the largest integer storable in a computer.

1) Ordinarily at the end of a pass we would need to travel through the entire pattern deleting all flagged points in order to prepare the pattern for the next pass. But with our labeling technique, the flagged points which are candidates for deletion have the label i-MAXINT in pass i. In pass $i + 1$, the value i-MAXINT becomes a threshold, and all points with labels less than or equal to this threshold are considered white. The flagged points are thus only conceptually deleted, and it results in speeding up the algorithm.

2) This labeling technique gave our algorithm reconstructability. We have adopted the reconstruction procedure as given in [11]. In essence, the reconstruction tech-

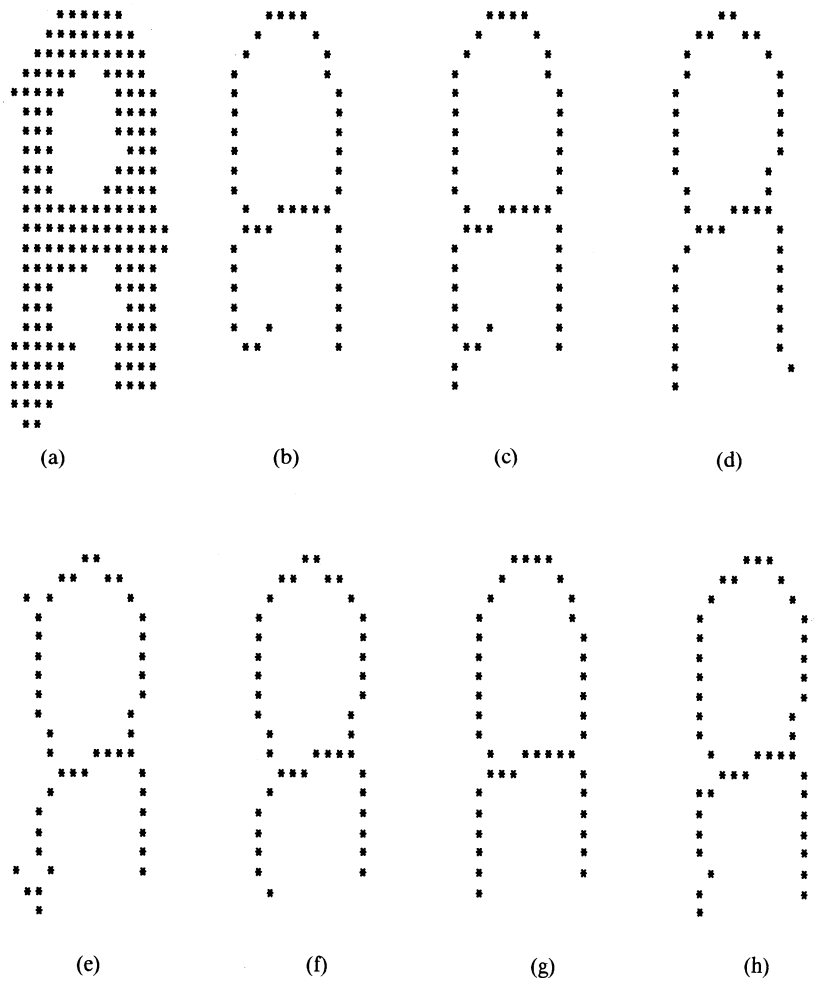

Fig. 7. A specimen of an original pattern, and the skeletons obtained by applying the 15 algorithms over that pattern. '*' indicates dark points. The meaning of the abbreviations below each of the skeletons is given in Section III. (a) A Specimen Pattern. (b) ARCL. (c) BELA. (d) BEUO. (e) BEUS. (f) HILD. (g) PAVC. (h) PAVB. (i) PAVR. (j) SRVH. (k) ST4S. (l) ST2S. (m) TM4F. (n) TM4E. (o) TM2E. (p) SPTA.

nique is the following: given an input skeleton of labeled dark points, it iteratively pads dark points around the skeleton. The number of dark points padded around a point p of the skeleton is a function of the value of the label of p. For the sake of completeness, the reconstruction algorithm is formally described in Appendix II.

V. Experimental Results and Discussion

To compare the average performance of the SPTA to the 14 skeletonization algorithms reviewed in Section II, we tested all the algorithms on a data set of 648 hand-printed characters (letters "A" to "Z" and numerals "0" to "9"). The average size of the binarily digitized characters was 23 rows and 17 columns, with the maximum size being 40 rows and 35 columns. The characters were handprinted by different students of Concordia University, Montreal, and were digitized by an ECRM 5200 auto-reader. The algorithms were coded in PASCAL Version 3.1 and run on a Control Data Cyber 170-825 computer, the coding being as intuitively efficient as possible. If any of the algorithms required the input pattern to be smoothed, we adopted the smoothing techniques as described in [6], [17]. Since these techniques are well-known, we do not discuss their details. In our implementation, the SPTA followed a row-wise sequence, exactly as described in Appendix I.

To give an intuitive feeling on the types of skeletons produced by the different algorithms, Fig. 7(a) shows a specimen pattern from our data base. For this pattern, the skeletons produced by the 14 algorithms (reviewed in Section II) and by our SPTA, are shown in Figs. 7(b) to 7(p). In Table I we give for our data base of 648 patterns and for each of the 15 algorithms the following information: the average cpu time it took to skeletonize a pattern, the average number of passes required to skeletonize a pattern, the average cpu time required per pass, the connectedness (eight-connected or four-connected) of the skeletons produced, and whether the algorithm necessitated that the input pattern be smoothed. If smoothing was necessitated, we have included the smoothing time in the average cpu-time required to skeletonize a pattern.

We notice from Table I that about half the number of algorithms necessitated smoothed patterns as input. The three algorithms SRVH, SR4S and TM4F produced four-connected skeletons; all others produced eight-connected skeletons. Usually the algorithms require two to four passes to complete the skeletonization. If we are to rank the algorithms in ascending order of cpu time required for

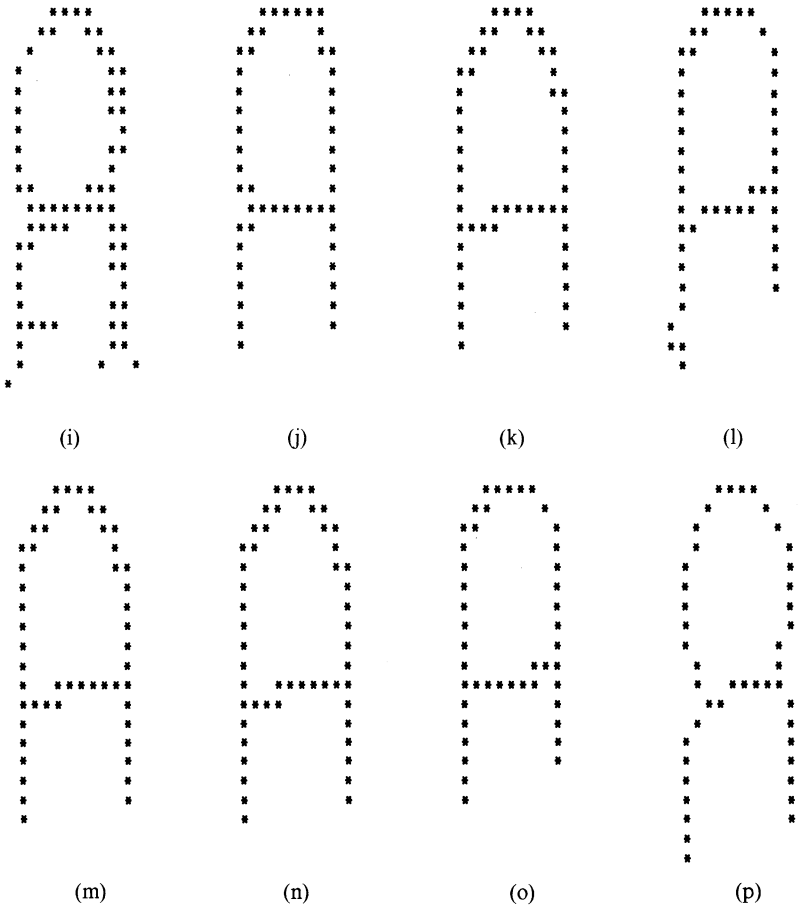

Fig. 7. (*Continued*)

TABLE I
EXPERIMENTAL RESULTS OF ALL 15 SKELETONIZATION ALGORITHMS.
(648 patterns were skeletonized by each algorithm.)

ALGORITHM	AVG CPU-TIME PER PATTERN	AVG NUMBER OF PASSES	CPU-TIME PER PASS	CONNECTEDNESS OF PATTERNS	SMOOTHING (*)
ARCL	390.61 MS	2.87	136.10	EIGHT	NO
BELA	392.08 MS	2.90	135.20	EIGHT	NO
BEUO	451.90 MS	3.46	118.43	EIGHT	YES
BEUS	416.40 MS	3.27	114.46	EIGHT	YES
HILD	393.12 MS	3.32	118.41	EIGHT	NO
PAVC	186.22 MS	2.93	63.55	EIGHT	NO
PAVB	319.34 MS	3.91	81.67	EIGHT	NO
PAVR	224.04 MS	2.57	87.18	EIGHT	NO
SRVH	513.02 MS	4.33	118.48	FOUR	NO
SR4S	392.79 MS	2.23	157.25	FOUR	YES
SR2S	421.26 MS	3.00	126.38	EIGHT	YES
TM4F	401.07 MS	2.24	160.24	FOUR	YES
TM4E	396.13 MS	2.23	158.74	EIGHT	YES
TM2E	431.77 MS	3.04	128.17	EIGHT	YES
SPTA	183.75 MS	3.33	42.53	EIGHT	YES

*This column indicates whether smoothing was required by the algorithm before skeletonizing the pattern. Whenever smoothing was required, the column indicating the average cpu-time per pattern included smoothing time. For our data base of 648 patterns, smoothing time was found on an average to be 42. 13 ms per pattern.

ARCL Arcelli's algorithm [1].
BELA Bel-Lan's and Montoto's algorithm [2].
BEUO Beun's algorithm using orthogonal neighbors [3].
BEUS Beun's algorithm using Saraga–Woollon's criteria [3].
HILD Hilditch's algorithm [8].
PAVC Pavlidis' classical algorithm [11].
PAVB Pavlidis' basic algorithm [11].
PAVR Pavlidis' reconstructability algorithm [11].
SRVH Stefanelli–Rosenfeld's version of Hilditch's algorithm [15].
SR4S Stefanelli–Rosenfeld's algorithm with four scans per pass [13], [15].
SR2S Stefanelli–Rosenfeld's algorithm with two scans per pass [13], [15].
TM4F Tamura's algorithm ensuring four-connectedness [16].
TM4E Tamura's algorithm ensuring eight-connectedness [16].
TM2E Tamura's algorithm with two scans per pass [16].
SPTA The safe-point thinning algorithm proposed by the authors of this paper.

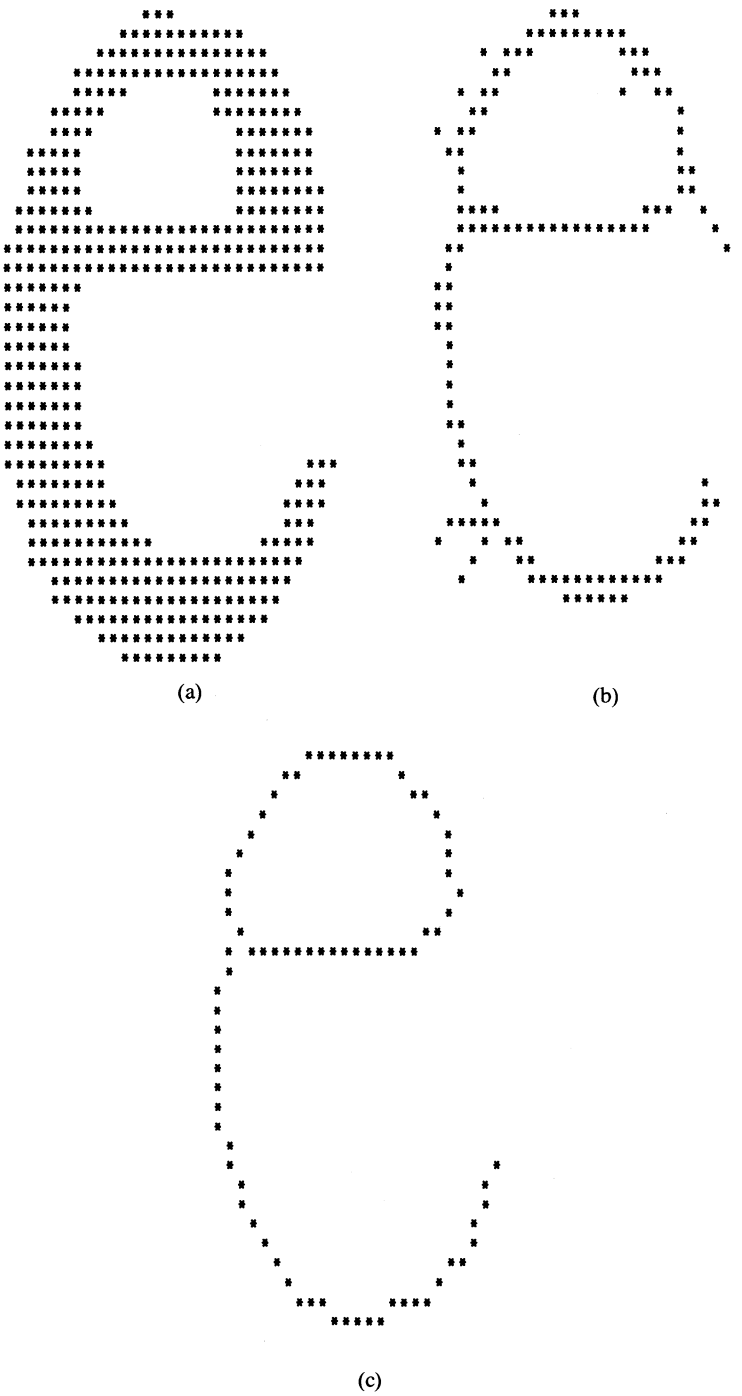

Fig. 8. A sample pattern and its skeletons produced by PAVR and SPTA. (a) A Specimen Pattern. This pattern was not from our data base, but was identical to a figure given in Pavlidis [12]. (b) The Skeleton Produced by PAVR. It is identical to the one obtained by Pavlidis [12]. (c) The Skeleton Produced by SPTA.

skeletonization, we get the following sequence: SPTA, PAVC, PAVR, PAVB, ARCL, BELA, SR4S, HILD, TM4E, TM4F, BEUS, SR2S, TM2E, BEUO, SRVH. In passing we wish to make one comment: algorithm HILD is faster than algorithm SRVH; i.e., the algorithm originally proposed by Hilditch in [8] is faster than an algorithm [15] that Stefanelli and Rosenfeld called a "simplified version" of Hilditch's algorithm. A detailed analysis of HILD versus SRVH appears in [10].

From above we see that SPTA is the fastest. The only algorithm as fast as SPTA is PAVC. However, PAVC does not have the reconstructability, whereas SPTA does. The only other algorithm which has reconstructability is PAVR, but it is slower than SPTA. To compare the reconstructability of SPTA with PAVR, we implemented the reconstruction algorithm as given in Appendix II. Testing on our data base of 648 patterns, we observe the following: when the reconstructed pattern was superimposed on the original pattern, the SPTA on an average gave a point-to-point match of 94 percent, whereas PAVR gave a match of 100 percent. Thus we concede that PAVR has a slightly better reconstructability than SPTA. In fact Pavlidis, the designer of PAVR, proves in [11] that his algorithm would always give 100 percent reconstructability. However, we do wish

to compare the quality of the skeletons produced by PAVR to the quality of the skeletons produced by SPTA. Fig. 8 shows the skeletons produced by PAVR and SPTA, of another specimen pattern. Both Figs. 7(i) and 8(b) show that PAVR produces a skeleton with spurious tails, and that some strokes in the skeleton may not be thinned to a line of width one. We believe that it is a generally accepted principle that the presence of spurious tails and thick strokes in the skeletons causes a deterioration in the quality of the skeletons. Taking this into consideration, we can say that the skeletons produced by SPTA are superior to those produced by PAVR.

Thus we can conclude that our SPTA is the fastest of the 15 algorithms we have tested, it produces good quality skeletons, and it has good reconstructability.

V. Conclusion

We have reviewed 14 skeletonization algorithms developed by other researchers. We have proposed a new algorithm called SPTA, and on a large data base of 648 patterns we have experimentally compared it with the reviewed algorithms. Our SPTA is shown to be the fastest. It produces good quality skeletons, and it has good reconstructability. The only other algorithm that is as fast as our SPTA is Pavlidis' PAVC [11]; however, PAVC does not have reconstructability. Pavlidis' other algorithm PAVR [11] does have reconstructability, but PAVR is slower than our SPTA. Moreover, the skeletons produced by PAVR are found to have spurious tails, and the strokes of the skeletons may have a thickness of more than one. Thus in conclusion we can say that as compared to the other known skeletonization algorithms, SPTA is optimal; i.e., it is the fastest, it produces good skeletons, and it has robust reconstructability.

Appendix I

Here we show the Pascal-like formal algorithmic version of our SPTA. The patterns can be scanned either row-wise or columnwise. The choice of the scanning sequence may lead to different skeletons; however, a skeleton may not be unique [4].

```
{ ··· Declaration of global variables ··· }
    type PATTERN-TYPE : array [1..HEIGHT, 1..WIDTH]
                        of integer;
                        {type of the pattern, where
                        HEIGHT and WIDTH are its di-
                        mensions}
                    i: integer; {i represents the iter-
                                 ation number during
                                 the skeletonization
                                 process}

{The following procedure drives the scanning of the pat-
                        tern}
procedure DRIVER (var PATTERN: PATTERN-TYPE);
var  j : [0,2];
     { j indicates the 'type' of scanning;
       j = 0 ⇒ detection of right and left safe-points,
       j = 2 ⇒ detection of top and bottom safe-points}
begin
     i := 0;
     repeat
       i := i + 1;
       for all j's do
       SKELETONIZE (PATTERN, j); {procedure
                        SKELETONIZE is shown below}
     until NO-MORE;
     {NO-MORE returns TRUE when there are no
     more points to be flagged in the pattern}
end; {DRIVER}

{The following procedure performs one scan of skeletoni-
zation over the pattern}

procedure SKELETONIZE
  (var PATTERN: PATTERN-TYPE; j: integer);

var ROW, COLUMN : integer;
    p : integer; {point to be examined}
begin
  1  for all ROWS and all COLUMNS do
     {scan PATTERN with type j}
  2  begin p := PATTERN [ROW, COLUMN];
     {get a point p}
  3  if DARK (p) then
       {if p is dark and unflagged}
  4    if EDGE-POINT (p, n_j, n_{j+4}) then
         {if p is an edge-point, i.e.,
         it has n_j or n_{j+4} as white point}
  5      if not SAFE-POINT (p, j, j+4) then
           {if p does not satisfy either
           Boolean expression S_j or S_{j+4}}
  6        FLAG (p, i)
           {flag the point p; procedure shown below}
  7      else { p is a safe-point}
           LABEL (p, i);
           {label the point p; procedure shown below}
     end; {of for-loop}
  end; {SKELETONIZE}

{The following procedures FLAG and LABEL are called by
procedure SKELETONIZE}

{Procedure FLAG will flag a dark point p}
procedure FLAG (var p: integer; i: integer);

begin
  p := i − MAXINT;
end; {FLAG}

{Procedure LABEL will label a safe-point p}
procedure LABEL (var p: integer; i: integer);

begin
  if p is not yet labeled as safe-point then
    p := i;
end; {LABEL}
```

APPENDIX II

We show below the reconstruction algorithm as described by Pavlidis in [11]. This algorithm was used to reconstruct the skeletons obtained by our SPTA.

```
{let PATTERN be the matrix containing the pattern}
procedure RECONSTRUCT;

var p    : integer; {the considered point}
    labl : integer; {the label of that point}

  begin
    labl := highest label in the pattern;
    while labl > 1 do
      begin
        for all ROWS and COLUMNS do
          begin
            p := PATTERN [ROW, COLUMN];
              if p = labl then
                for all four-neighbors (n) of p do
                  if n < zero then
                    n := labl - 1;
              end;
        labl := labl - 1;
      end; {while}

  end; {RECONSTRUCT}
```

ACKNOWLEDGMENT

During the various stages in the development and refinement of our SPTA, we had discussions with Mr. Michael R. Payette of Concordia University, Montreal, Canada. The issues he raised often helped us in our development process. We thank him for his contribution.

REFERENCES

[1] C. Arcelli, "A condition for digital points removal," *Signal Processing*, vol. 1, pp. 283–285, 1979.
[2] A. Bel-Lan and L. Montoto, "A thinning transform for digital images," *Signal Processing*, vol. 3, pp. 37–47, 1981.
[3] M. Beun, "A flexible method for automatic reading of hand-written numerals," *Phillips Tech. Rev.*, vol. 31, no 4, pp. 89–101, pp. 130–137, 1973.
[4] E. R. Davies and A. P. N. Plummer, "Thinning algorithm, A critique and a new methodology," *Pattern Recognition*, vol. 14, pp. 53–63, 1981.
[5] E. S. Deutsch, "Thinning algorithms on rectangular, hexagonal and triangular arrays," *Commun. Ass. Comput. Mach.*, vol. 15, no 9, pp. 827–837, Sept. 1972.
[6] W. Doyle, "Recognition of sloppy hand printed characters," in *Proc. of West Joint Comp. Conf.*, May 1960, vol. 17, pp. 133–142.
[7] A. Gudsen, "A quantitative analysis of preprocessing techniques for the recognition of hand-printed characters," *Pattern Recognition*, vol 8, pp. 219–227, 1976.
[8] C. J. Hilditch, "Linear skeletons from square cupboards," *Machine Intelligence*, vol 4, pp. 403–420, 1969.
[9] B. Moayer and K. S. Fu, "A tree system approach for fingerprint pattern recognition," *IEEE Trans. Comput.*, vol. C-25, pp. 262–275, 1976.
[10] N. J. Naccache and R. Shinghal, "An investigation into the skeletonization approach of Hilditch," *Pattern Recognition*, in press.
[11] T. Pavlidis, *Algorithms For Graphics And Image Processing.* Rockville MD: Computer Science Press, 1982, pp. 195–214.
[12] T. Pavlidis, "A flexible parallel thinning algorithm," in *IEEE Proc. Conf. on Pattern Recognition and Image Processing*, Dallas, TX, Aug. 1981, pp. 162–167.
[13] A. Rosenfeld, "A characterization of parallel thinning algorithms," *Inform. Contr.*, vol. 29, pp. 286–291, 1975.
[14] P. Saraga and D. J. Woollons, "The design of operators for pattern processing," in *IEE NPL Conf. on Pattern Recognition*, Teddington, 1968, pp. 106–116.
[15] R. Stefanelli and A. Rosenfeld, "Some parallel thinning algorithms for digital pictures," *J. Ass. Comput. Mach.* vol 18, no 2, pp. 255–264, 1971.
[16] H. Tamura, "A comparison of line thinning algorithms from digital geometry viewpoint," in *Proc. 4th Int. Joint Conf. on Pattern Recognition*, Kyoto, Japan, 1978, pp. 715–719.
[17] S. H. Unger, "Pattern detection and recognition," *Proc. IRE*, pp. 1737–1752, Oct. 1959.