A Segmentation and Object Extraction Algorithm with Linear Memory and Time Constraints

R.S. Anbalagan

Innovision Corporation
3201 Latham Drive
Madison, WI 53713

G. Hu

A.K. Jain

Michigan State University
A714 Wells Hall
East Lansing, MI 48824

ABSTRACT

An experimental segmentation and object extraction algorithm is described. The system has been developed for medical image processing with the primary application to DNA (deoxyribonucleic acid) sequencing. A typical DNA sequencing may involve processing the image of an autoradiogram of size 14 x 17 inches resulting in 2048 x 8600 digitized image under the specified spatial resolutions. The digitized image is too big to manage even using super-mini computers such as DEC VAX 11/780, and to perform any amount of classical image processing. Therefore, an elegant hardware and software design is necessary to deal with the large image and to complete the image understanding task in an efficient manner. Data acquisition, image processing, object recognition and a contextual classification, and sequence generation are four of the major tasks performed by the system in a tightly coupled pipeline mode of operation. Each phase utilizes the data from its predecessor, performs the relevant task when the required information for decision is available, usually a few rows of the input image, and passes the data to the successor for further processing, all on the fly. This paper focuses on the Image processing aspects of the system, and describes the run-length image representation, a linked list data structure, a heuristic connected component analysis algorithm based on the data structure, a primitive object segmentation algorithm and feature extraction.

1 Introduction

Two dimensional image processing leading to object recognition has found many applications. Industrial parts recognition for robot vision; blood cell and chromosome recognition for medical diagnosis; satellite image processing for urban planning, or weather forecasting; numerous quality control applications in food and manufacturing industries, etc. are some of the applications. A problem of current interest in biomedical image processing and the topic of description in this paper is Automated DNA Sequence Film Reading [1, 4]. Figure 1.a shows an ideal picture of a DNA sequence film. However, in practice, the X-ray film obtained and to be processed is shown in Figure 1.b. The objective of the reading of the film is to to enumerate the ordered sequence of neucleotides as determined by the presence of dark bands in the film.

We have developed a DNA sequence reader, an expert vision system that accepts autoradiograms as shown in figure 1.b, and enumerates the proper sequence of neucleotides to the researcher. The detection accuracy should be such that two objects (dark bands) must be resolvable if they are spatially separated by 0.25mm or more. This specification and the possible distortions that occur in practice dictate that the digitizing be done at 0.05 mm line spacing. An experimental film is typically 14" x 17", resulting in as many as 8600 scan lines. The pixel spacing is selected such that the 14" width is mapped onto a 2048 element CCD array with appropriate optical components. Thus, the digitized image of the autoradiogram film is 8600 x 2048 pixels. Besides the normal difficulties that an image processing system designer encounters, we have the additional challenge to deal with a large image format containing a few thousand objects to be recognized. The system also needs to be practical in processing speed and economical in cost to be useful.

Various stages of the system are designed for pipeline operation. The major constraint of the design is that at any point in time during processing, only a small number of scan lines (less than 100) of the digitized image are available for the algorithms. Interesting results from this design constraint are the segmentation and feature extraction algorithms that operate in linear time and memory requirements. These algorithms are the subject of discussion in the remainder of this paper.

2 Connected Component Extraction

Many connected component labeling algorithms have been successfully applied on binary images as a first step towards shape recognition. The classical approach [6] includes two passes over the image data, namely:

Pass 1: For each BLACK pixel p of value 1 (object), assign a label to its scanned (4- or 8- connected) neighbors. If none of the pixel p's neighbors has been labeled, a new label is issued to p. An "equivalence" table is constructed as the scanning proceeds, which is the key to a connected component labeling algorithm, necessary when two components with different labels meet together at some point.

Pass 2: Label each pixel of the original image according to its assigned label in the equivalence table.

Pass 2 often serves the purpose of displaying the results of pass 1, and hence testing the correctness of label assignment and construction of equivalence table. Many applications do not care for the labels themselves, but demand a proper representation of the connected components in a manner such that they can be easily reconstructed when needed. Thus the labeling part (pass 2) can be removed from the algorithm. Consequently, an explicit equivalence table construction is redundant and can be eliminated. Merging of two equivalent labels can be accomplished during the scan using an appropriate data structure. By the removal of equivalence table and pass-2 from the classical algorithms, the technique becomes a 1-pass connected component extraction algorithm.

The only assumption that limits the generality of our algorithm is that there will be no explicit labeled image. This assumption often holds true in most applications where connected component labeling is only an early step in an entire shape

recognition process. The proposed algorithm, however, is very general in the following:

1. The algorithm does row processing in raster scanning order, i.e., only one row of the image is actually required and stored in internal memory at any time. This is important in situations where the input images are unusually large and real time response is required.
2. Every connected component represented in the data structure of the algorithm can be reconstructed if necessary.
3. The technique can be easily extended to more than two dimensions.
4. For a M x N image, the internal storage requirement as well as the overall processing time is O(N). This particular characteristic makes the proposed algorithm very effective for large images requiring only limited computational resources.

2.1 Data Structure

The data structure is essentially a method for representation of objects in an image. Many image representation techniques have been proposed and successfully used in the past, from 2-dimensional array to chain code, from run-length to quadtree, each having its own strengths and weaknesses. Due to the one-row-at-a-time processing requirement and the available storage, 2-dimensional array and the quadtree methods are not feasible. Chain code representation requires an explicit boundary of the object and, therefore, unsuited for the problem at hand. A modified run length representation is suitable for the image.

A run is usually defined as a pair (s,l) where,

s = (x,y) is the coordinate of the starting point of a sequence of consecutive 1's (object) in some pre-specified direction

l is the length of the sequence (the total number of consecutive 1's in the specified direction).

We prefer the run length representation along the column direction as opposed to that along the conventional row/line direction. The choice is based on the fact that all the objects in the image are oriented in the horizontal direction and we wish to extract the principal axes of the objects which are readily formed by the mid-points of the column runs.

The data structure must also fulfill the tasks of keeping track of object labels and their equivalence classes. A connected component may have many (varying numbers of) runs bounded between 1 and N, suggesting that a linked list structure be used for the representation of the object. The component label information is kept in the header of the linked list which is a member of yet another linked list that keeps track of all the connected components. Finally, a pointer is added to the run length representation to preserve adjacency relation between successive columns. In short, a run is represented by a triplet (s,l,p), where:

s, l are defined earlier, and

p is a pointer to the next run belonging to the same component.

A simple example, Figure 2, illustrates the above representation. Figure 2 shows the two connected components and the corresponding linked list representations.

Two one-dimensional arrays of size N (for M x N images) are used to decide the equivalence relation on the components. One of the arrays, ACCUM, accumulates lengths of current runs, where current runs are the runs at columns where 1 appears at the time when the current row is being scanned. The other array, LABEL, keeps track of component labels to which the current runs belong. For example, in Figure 2, while processing the pixel pointed by the arrow, the two arrays appear as:

ACCUM: 0 2 0 2 2 3 0 2 1 0

LABEL: 0 1 -1 1 1 1 0 2 2 0

When a linked list is completed , i.e., a connected component is closed, it is passed on to the segmentation and feature extraction process. The storage for the list is freed for other use. The maximum number of linked lists (connected components) which can be "active" simultaneously is N/2 [8], and hence the number of header pointers is bounded by N/2. The total number of nodes in all active linked lists is bounded by cN, where c is a small positive constant. Since N is the total number of columns, no more than N columns can be seen at the same time. If no multiple runs in the same column appear, then c is 1. The worst case c is bounded by N/2, though c is found to be between 1 and 2.

The total storage required is the sum of storage for image and the data structure. The storage required for image is 3N (accumulators ACCUM and LABEL, and the current row of input image). The storage for active connected component is bounded by 0.5N. The nodes in the active linked lists are bounded by kcN, where k is a constant depending on how many fields are defined in each node (triplet in our case) and c as described before. The total memory required is:

$$3N + 0.5N + kcN$$

Thus the amount of internal storage is O(N).

2.2 Connected Component Algorithm

The connected component algorithm described below is based on the following principles:

1. A run starts at a 0-1 transition;
2. A run ends at an 1-0 transition;
3. Merge can occur only when a new run starts;
4. Any two runs of two disconnected components have at least one '0' pixel everywhere between them as disconnecting gaps.

The accumulator array ACCUM serves two purposes:

1. accumulate run lengths of each active column runs;
2. indicate presence or absence of an active run at a column .

Initially both the arrays ACCUM and LABEL are initialized to zero. At a row r, a left to right raster scan is performed. A pixel p at column c, i.e., pixel (r,c) of the image, may signal one of the following transitions based on the value pair (ACCUM(c), p):

0-0: pixel p continues to be in the background;
0-1: pixel p belongs to a boundary of a component and initiates a new run;
1-1: pixel p belongs to the interior of a component;
1-0: pixel p belongs to the background, pixel (r-1,c) belongs to the boundary, and the active run at column c terminates.

The key steps of the algorithm are outlined below:

1. Scan incoming row r of the image stored in the BUFFER;
2. For each pixel p in the BUFFER at offset c, i.e., pixel (r,c), determine the type of the transition as defined above;

3. switch (transition){
case 0-0: Reset LABEL[c] to zero; break;
case 1-1: Increment the run length counter ACCUM[c] by 1; break;

case 0-1: Increment the run length counter ACCUM[c] by 1;
Assign a label to LABEL[c] according to the first non-zero value from ABS(LABEL[c-1]) or ABS(LABEL[c+1]), in that order. If both are zero, assign a new label to LABEL[c];
Allocate space for a node triplet (s,length,next) for the new run; define s as (r,c); insert the node to the linked list pointed by the head[LABEL[c]];
If the two linked lists indexed by LABEL[c-1] and LABEL[c+1] are non zero, then merge the two indicating as a single component. Change all LABEL[i]'s that have the same label as LABEL[c+1] to LABEL [c-1]. This corresponds to the equivalence table mechanism in other labeling algorithms. Here we actually merge the two pieces of a component rather than recording equivalent labels. break;
case 1-0: Enter the length as defined by the value in ACCUM[c] in the field corresponding to the node triplet in the linked list; reset ACCUM[c] to zero;
If the component is closed reset the LABEL[c] to zero, otherwise negate the value in LABEL[c] (this action is necessary to preserve 8-connectedness of the objects in the image);
Check if the component is completed; (this can be done by defining a run count for each component list with initial value 0, increment the count by 1 for each 0-1 transition, decrement by 1 for each 1-0 transition, and signal 'completed' when the count vanishes to zero.)
If the connected component is closed, the corresponding linked list is passed on to the next stage in the pipeline for further processing such as segmentation and feature extraction;break;
}
4. go to 1;

The example in Figure 3 illustrates the operation of this algorithm.

3 Knowledge-Based Image Segmentation

As a result of the connected component extraction algorithm described in the previous section, each connected component is represented as a linked list whose nodes are column runs of the component. Classical segmentation and feature extraction algorithms fail to perform on the chosen data structure. Segmentation algorithms often require a prior knowledge about objects of interest, for example shape, size or boundary density characteristics and perform best when such information is readily available. In our test images, shape of the objects (dark bands) are well defined and could be used as a criteria during segmentation. Shape can be easily characterized by the boundary points of the objects under study. Many boundary based object segmentation algorithms can be found in the literature[2,6,7]. A popular segmentation algorithm[8] based on the boundary points uses curvature chain and smoothing filters. Large pieces of connected components are split at points of high connectivity, small pieces are merged if they are in close proximity and satisfy certain merging criteria, or a part of the boundary can be matched against a preconstructed model. In our run length representation, explicit boundary information (for example, the chain code) is not available and reconstructing the complete object is certainly out of question.

We propose the following knowledge-based segmentation algorithm for the objects in the images. The basic characteristics of the objects of interest are rectangular shape, horizontal orientation, and tilts of less than 45 degrees. The widths and heights of the rectangular objects are bounded by some values that are locally constrained and determined as the processing continues (Figure 1).

The segmentation algorithm consists of the following four key steps:

a. For each connected component, decide if it needs to be split according to the prior knowledge about the geometrical properties of objects. Dramatic increase in area, increase in height-width ratio, etc. are some examples of decision criteria.
b. For those components requiring split, find splitting points which are points of large concavity.
c. Merge small components using criteria such as closeness, collinearity, etc.
d. Remove all small components having area below certain threshold and not belonging to any object.

Each of the above steps require further explanation.

a. Selecting components for splitting:

Area too large: Area of a component is readily computed to be the sum of the lengths of its column runs in the linked list. The threshold for 'Too large' is determined either by using a global maximum value or by using the localized constraint as defined by the statistics based on already segmented objects.

Abnormal height-width ratio: Height and width of a component can be estimated by an enclosing rectangle or very simply by using the maximum length of runs as height and the difference between the first and last column ordinates as width. Abnormal is defined by a global maximum and by local statistics based on segmented objects.

Irregular boundary: Smoothness is measured by the connectivity of the midpoints of the runs of a component. A component with smooth boundary has its runs having 8 adjacent mid points. If there is a sharp change in x-coordinates of midpoints, we know that the boundary is irregular. 'Sharp' is again defined using a threshold which may be an absolute value (3 pixels, for example) or as a ratio of change in x-coordinates of midpoints to the object's height.

b. Locating points of concavity:

Although we do not have explicit boundary information in our linked lists, boundaries are implicitly embedded in starting points of runs and run lengths. If a boundary is not smooth, i.e., it has points of sharp concavity or convex points, the x- coordinates of the starting points or ending points of the runs will show large discontinuity as the y- coordinate is traversed. For those connected components that need to be split, a difference operator is applied to the localized average of x-coordinates belonging to either sides of, say, column c. The difference D provides the degree of disconnectivity at column c. A second order difference is computed using D[c+1] and D[c]. A 'zero crossing' from a positive to negative value suggests a concave point. The concave point with the maximum difference value is chosen as a candidate for split. The same procedure is repeated over the end points of the runs. Similar principle has been widely used in edge detection algorithms.

The above procedure works as long as there are no multiple runs in the same column belonging to the same object as shown in Figure 4. In situations like this, the starting points and ending points of some columns are not unique. A recursive split procedure as defined below deals with this anomaly:

> Form a sub list collecting one run per column, order is not important. Locate the candidates for splitting as described above. Perform the split in the parent list. Repeat the procedure until no further splitting is possible.

c. Splitting:

The two candidate points on the boundary of a connected component, if present, are used to make a cut to remove either the touching object or a noisy fragment of an object. The process is illustrated for splitting a component of two overlapped objects in Figure 5-a. However, it is not always possible to merely use the line joining the two concave points as a cut line as illustrated by a dotted line in Figure 5-b. Sometimes, only one concave point is available. A heuristic approach is employed in this experiment employing the a priori knowledge. For each candidate point p, locate a set of boundary points having the same x or y coordinate as the point p. The set may have at most four members. Together with p, we may have five boundary pixels as candidates for end points for cut. The boundary point having the 'shortest distance' to p is the matching end point defining the cut line. In figure 5-b, the solid lines are the cut lines determined by this algorithm.

d. Merging:

A single object may be broken into many pieces due to improper thresholding, poor quality of input image or due to other preprocessing operations. It is essential that the algorithm detects such cases and merges them as a single object. Conventional wisdom such as 'object is too small' and 'object is close to another' is employed to flag situations for merging. However, a priori knowledge about the object model being rectangular suggests that the midpoints of runs be collinear or curvilinear for the two objects eligible for a merge. The merging of two objects is easily accomplished by combining their corresponding linked lists.

4 Feature Extraction

On each of the isolated objects, a set of basic geometric features are computed to assign a decision label. The decision process by itself is an Expert System and is discussed elsewhere. Some of the useful features are:

- Area
- Perimeter
- Approximate bounding rectangle
- Centroid
- Principal axis
- Width & Height
- Orientation or slope
- Density
- Distance to its neighbors

All of the above features, with the exception of density, are computed using the linked list representation of the object. The classical computation methods for the above features often require memory and time bounded by O(N^2). Our algorithm computes the above features in linear time requiring only O(N) memory space.

5 Discussion

We have presented a 2D shape analysis system for object segmentation and feature extraction for automatic enumeration of DNA sequences seen in autoradiograms. Our approach uses run-length to represent binary images and linked list data structure for connected components and isolated objects. Split and merge operations as well as feature extraction are performed using the same data structure. The contributions of our approach are threefold: First, the system is general enough to handle images of any size M x N, as long as cN (c is usually less than 2) does not exceed internal storage of the system. The processing speed of the algorithm and the total storage requirement are both O(N). Second, the system uses an improved one-pass connected component algorithm that extracts objects in an efficient manner without the use of equivalence table making it a powerful technique for real-time processing applications employing raster scanning methods. Finally, we have shown that the use of a priori knowledge and the specific domain knowledge can help in the design of complex segmentation procedures rather efficiently. Experiments on a set of 20 real images have shown that the algorithm is robust and extracted 90% or more objects without any further processing. The algorithm is implemented on a VAX 11/780 running VMS operating system. Empirical data suggest that a 8K x 2K image having approximately 4000 or more individual band objects can be processed in less than an hour (major portion of the time is spent in image acquisition). Improvements to the algorithm as well as implementation of it on a stand alone system based on M68020 microprocessor are in progress.

6 References

[1] Anbalagan, R.S., Warner, M., Avdalovic, N., Summers, M. and Wilding, P. "Automated DNA Sequence Film Reading", ASCB'86, Jun 2-6, Washington D.C.

[2] Duda, R. and Hart, P. , Pattern Classification and Scene Analysis, John Wiley & Sons, 1973.

[3] Eccles, M.J., McQueen, M.P.C. and Rosen, D. "Analysis of the Digitized boundaries of Planar Objects" Pattern Recognition, Vol. 9, pp 31-41, 1977.

[4] Friedland, P. and Kedes, L.H. "Discovering the secrets of DNA", CACM, Vol. 28, No. 11, pp 1164-1186, 1985.

[5] Liu, R.H.C. Shape Descriptions and characterization of continuous Change, Ph.D. Thesis, Dept of Computer Science, SUNY at Stony Brook, 1976.

[6] Pavlidis, T. Algorithms for Graphics and Image Processing, Computer Science Press, 1982.

[7] Rosenfeld, A. and Kak, A.C. Digital Picture Processing, Academic Press, 1976.

[8] Samet, H. and Tamminen, M. "An Improved Approach to Connected Component Labeling of Images", IEEE CVPR'86, June 22-26, Miami Beach, FL, pp. 312-318.

Figure 1.a: Ideal picture of a DNA Seaquence Autcradiogram

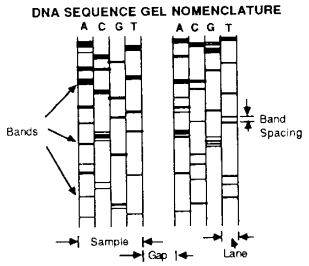


Figure 1.b: Typical Image of a DNA Seaquence Autoradiogram

Figure 2: Two Connected Components and the Corresponding Data Structures

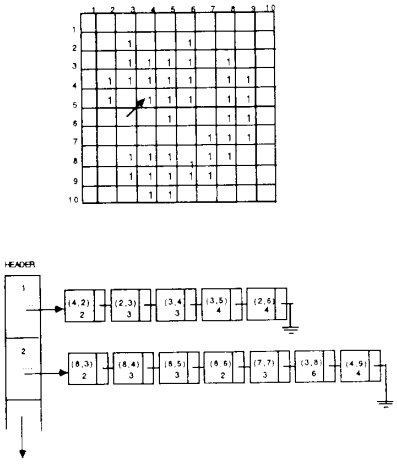
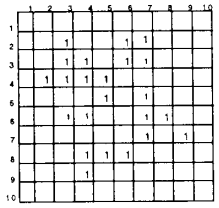


Figure 3: Illustration of Connected Component Analysis Algorithm



(r,c)	ACCUM	LABEL	Linked Lists
3,7	0021022000	0011022000	Lists 1 and 2 -- two components
4,5	0132122000	0111111000	Merge as List-1 -- single component
4,7	0132100000	0111111000	
6,8	0011002100	0011002200	
7,4	0000002100	0000002200	(Object-1) List-1 completed
9,1	0002120000	0002200000	
9,6	0002000000	0002000000	
10,4	0000000000	0000000000	(Object-2) List-2 completed

Figure 4: Recursive Split for extracting overlapped objects

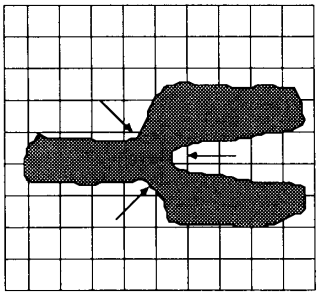


Figure 5.a: Normal Split

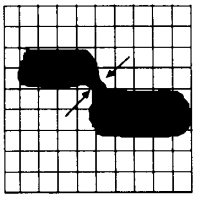


Figure 5.b: Heuristic Split