Chapter 1

INTRODUCTION

§1.1 The Problem and Possible Approaches

Separation of objects from their background is a major problem in pattern recognition and scene analysis [3], [9]. In the case of grey scale pictures, it is commonly referred to as segmentation. In the simpler case of binary pictures, it is going by the name of "*detection of connected components*". It is the latter case which is adressed in this report. An algorithm to perform the detection of connected components is examined in great details, and a computer program is presented in Appendices.

Detection of connected components in binary pictures is an indispensable step in such applications as automatic visual inspection, optical character recognition, extraction of karyotypes from photomicrographs of mitotic cells, robot vision, fac-simile coding systems, etc.

The approach to the detection of connected components which is described here is appropriate when the size (in bits) of the picture, and the number of pictures to process prevent storing the whole image fields in permanent memory. Our work is based on the assumption that the picture is scanned in raster mode and only a small part of it is accessible concurrently. This requires that the

detection of connected components be achieved in *real-time*, and in a *sequential fashion.*

Within this framework, two possible approaches can be advocated. One is based on border finding techniques, see e.g., Cederberg [2]. According to Rosenfeld, the second approach which is adopted here and is based on tracking runs of the figure rather than its borders is the best choice for most purposes [9]. We shall have more to say on the appropriateness of these approaches later.

Some will object that sequential operations are "slow", and connected components can be counted and even labelled at much higher speed using parallel (cellular) array processors [5],[6],[8]. In actual fact, the reasons why parallel operations were excluded from consideration here are twofold. In the first place, the largest cellular arrays to date consist of up to 128 × 128 processors [7], and this is still much too small for the kind of pictures we have in mind. For larger pictures, the processors have to be applied blockwise. In the second place, in existing cellular arrays, each cell has very little memory (up to 128 bytes) [8]. This sole constraint would place unbearable restrictions on the kind of processing we wish to perform.

With this alternative settled, we shall say more, in Chapter 2, about what is really meant by "*real-time*" and "*sequential*" detection of connected components.

§1.2 Topological and Geometrical Information

In virtually every application, it is normally expected that the detection of any connected component be accompanied by some concise description of that component. Component description usually consist of information of both *topological* and *geometrical* nature.

Topological information is concerned with neighborhood and surrounding relations between connected components of the figure and its background. For instance, the number of pin-holes in some conducting track on a printed circuit board, or surrounding relations between such tracks represent information of a topological nature. Topological information may be represented by *adjacency-* or *neighborhood-trees* [1], which in turn can be coded by their *vertex-* or *edge-string* (see also Chapter 5). In the algorithm described here, the choice of the level of topological information desired is proposed as an (interactive) option in the computer program.

For what concerns geometrical information about connected components, we are confronted with a very wide spectrum of possible choices. At one end, one might be content with the sole chain code of the component's borders, or some equivalent representation of the component's edges, see [2] and [4]. This kind of representation is quite appropriate for data compression purposes [2]. However, from the viewpoint of pattern recognition and scene analysis, it does not reveal much of the "*structure*" of the connected components. In order to illustrate the other end of the spectrum, let us suggest one possible description of character M. In addition to the chain code of its border, we might wish to know its perimeter, area, center of inertia, moments, etc; we might also want to learn that it is made of four strokes, two vertical ones and two skew ones, and three 2-junctions; that two 2-junctions turn their skew angle downwards while the third one turns its upwards; that it has two end points, that these end points are the lower ends of the two vertical limbs, etc.

Evidently, this detailed geometrical description could be recovered, in a two-stage system, from the knowledge of the component's borders. However, such a processing would involve a significant duplication of effort with a concomitant waste of execution time and computing power.

The algorithm described here offers the possibility to go a long way towards obtaining—in real-time—a detailed geometrical description of connected components. As in the case of the topological information, however, the choice of the desired level of geometrical information is proposed as an interactive option. Let us then briefly turn our attention to these options.

§1.3 Program Options

Before we embark on a discussion of the two options offered in our program, a few words are in order about the hierarchical structure of entities handled by the algorithm.

At the lowest level, we are dealing with 1-dimensional structural elements, namely *runs*. Runs are described by the coordinates of their extremities and a few "run-parameters" characterizing adjacency relations between runs on successive rows.

At the next level, adjacent runs are assembled in 2-dimensional entities which we call *objects*. We distinguish essentially two classes of objects, namely,

blocks and *hinges* which correspond, roughly, to downstrokes and junctions of downstrokes respectively. Adjacency relations between objects—within a connected component—are described by "objects-parameters" which can be readily derived from the run-parameters corresponding to those runs the object is made of.

Finally connected components appear, at level 3 of hour hierarchy, as concatenations of adjacent objects. It is at that level that we shall come accross the discussion of our two options.

For what concerns the topological information, the point can be quickly made. The offered alternative is either to retain all of the topological structure of the image (this choice goes by the name of *full surrounding*), or to consider connected components of the figure taken in isolation, and to retain only the number of their holes, (this is called *restricted surrounding*). The former choice provides fairly exhaustive information. The latter is appropriate when we are dealing with collections of patterns which are simply connected.

It should come as no surprise that the decomposition of connected components into blocks and hinges—we shall make finer distinctions later, but there is no need to dwell on this at this stage—permits to lay bare or to make easily accessible much of the geometrical structure alluded to in the previous section. Thus, in terms of the geometrical information, the offered alternative is either to acquire and retain a small number of object-parameters which merely enable us to recover the edges of connected components, (this is, somehow, equivalent to chain-coding), or to acquire and retain a rich collection of object-parameters which reveal in an explicit way the adjacency relations and relative positions of objects within a connected components. These choices are called *restricted* and *full adjacency* respectively. There should be no need to insist that *full adjacency* is most appropriate for pattern recognition purposes.

We would like to stress that both *surrounding* and *adjacency* options are fully orthogonal in the sense that, everywhere in the computer program, they are implemented in an independent manner, hence any combination of both is readily feasible. Likewise, it would be a simple matter to condense the code of the program in order to optimize its performance for a given combination of choices.

§1.4 Summary of the Report

In Chapter 2, we make more precise the meaning of the words "*sequential*"

processing under "*limited memory*" and in "*real-time*". They represent the fundamental constraints we have to comply with in the sequel. We also introduce some fairly general terminology and notation.

Chapters 3 and 4 are concerned respectively with the first and second levels of our hierarchy of structural elements, *viz.*, runs, and objects. The organization of both chapters follows essentially the same pattern. Entities are formally defined together with the parameters required to represent their adjacency relations. We show how to compute these parameters and outline our real-time implementation of the computation.

In Chapter 5, we show how to recover connected components from the objects they are made of, and we discuss our *surrounding* and *adjacency* options. *Adjacency* comes first, in relation with edge-following in a figure (Section 5.2). Next, we introduce the *neighborhood-tree* as a representation for the topological information in Section 5.3 where we also suggest to represent that tree by its *edge-string*. There, we also formalize the concepts of *full* and *restricted surrounding*. Under appropriate—yet, fairly unrestrictive—assumptions, the detection of a connected component occurs at the time that its outer edge gets closed into a cycle. The discussion of this major event is the subject matter of Section 5.4.

In Chapter 6, we are concerned with the question of outputting the results of the program. We also display a sample output which illustrates the results obtained under each choice of the *adjacency* and *surrounding* options. Finally, Chapter 7 proposes an introduction to the program which can be found in appendices, and suggest some possible extensions.

The full program can be found in Appendices A through F. The code is written in Pascal. The program was tested on the VAX 11/780 computer at the Philips Research Laboratory, Brussels.

Acknowledgment: The authors would like to express their appreciation to M. Dekesel for his assistance with the implementation and testing of the program.

REFERENCES

[1] O.P. Buneman, "A grammar for the topological analysis of plane figures", in *Machine Intelligence 5*, B. Meltzer and D. Michie Eds., Edinburgh: University Press, 1969, pp. 383–393.

[2] R. Cederberg, *On the Coding, Processing, and Display of Binary Images,* Ph.D. Dissertation No. 57, Linköping Univ., Dept. Electr. Engrg., Linköping, Sweden, 1980.

[3] R.O. Duda, and P.E. Hart, *Pattern Classification and Scene Analysis,* New York: Wiley, 1973.

[4] B. Kruse, "A fast algorithm for segmentation of connected components in binary images", *Proc. First Scandinavian Conf. Image Analysis,* Lund, Sweden: Studentlitteratur, 1980, pp. 57–63.

[5] D. Nassimi and S. Sahni, "Finding connected components and connected ones on a mesh-connected parallel computer", *SIAM J. Comput.* vol. 9, pp. 744–757, Nov. 1980.

[6] S. Levialdi, "On shrinking binary picture patterns", *Comm. ACM,* vol. 15, pp. 7–10, 1972.

[7] J.L. Potter, "Image processing on the massively parallel processor", *Computer* vol. 16, pp. 62– 67, Jan 1983.

[8] A. Rosenfeld, "Parallel image processing using cellular arrays", *Computer,* vol. 16, pp. 14–20, Jan. 1983.

[9] A. Rosenfeld, and A.C. Kak, *Digital Picture Processing,* New York: Academic Press, 1976.