AUTOMATED CONTOUR LABELLING AND THE CONTOUR TREE

Joseph Roubal
Thomas K. Poiker
Simon Fraser University
Burnaby, B.C. V5A 1S6

BIOGRAPHICAL SKETCHES

Joseph Roubal received his B.A. (Cartography/Computer Science) from Western Washington University, and is currently an M.Sc. candidate (emphasis in computer mapping) at Simon Fraser University.

Thomas K. Poiker (formerly Peucker) studied at several West German universities, receiving his PhD (Geography) from the University of Heidelberg. He spent several post-doctoral periods at Harvard University and the Universities of Washington and Maryland. He is presently Professor of Geography and Computing Science at Simon Fraser University. He is also Affiliate Professor at the University of Washington, Editor-in-Chief of Geo-Processing (Elsevier, Amsterdam), principal of GeoProcessing Associates Inc., and an associate of Tomlinson Associates.

ABSTRACT

The contour tree is a construct (a graph) that represents the topological relations of contours on a map. An expansion of the concept allows the logical representation of contours, even if they are incomplete, i.e. have gaps or close outside the map borders. Using certain (usually obvious) assumptions about terrain, an algorithm has been developed that constructs the contour tree structure from line segments, identifies the gaps and labels the contours, given one reference elevation for one of the contour lines. The algorithm works successfully with gaps in up to 15% of the line segments. Wherever there is the possibility for error, the relevant line segments are indicated. Manual intervention, i.e. an operator indicating which segments are peaks, can increase the success rate significantly.

THE PURPOSE OF THE RESEARCH

The creation of digital elevation models and their incorporation into geographic information systems is an important task in many mapping agencies. Although there are many sources for the elevation information, the most complete sources are the contour lines on topographic maps. The ability to use these contours (often in the form of computer-scanned images) to extract the elevation information allows the use of this already existing source, minimizing the need for new collection efforts. Unfortunately, the process of extraction by manual methods is tiresome, painstaking work with much potential for errors of omission. This research is directed towards the partial automation of this task, using as little operator intervention as possible. A prototype computer program

has been developed which can identify the elevations of most contours in a partially automated system, and promises to be quite effective. This program recognizes the relationships between neighboring contours and utilizes the contour tree structure to establish topological relationships among the contour lines.

The contour tree

The contour tree is a powerful conceptual structure for representing the relationships among contour lines on a topographic map. It can be first found in the literature by Morse (1965, 1969) and has been expanded to a surface tree by Mark (1977).

Figure 1 is a simple contour map of Strange Island, B.C., and Table 1 is the contour tree representation of the contours that comprise the map. This example will be used throughout to illustrate the principles to be discussed. The root of the tree is the lowest contour line, the one that encloses all of the other contours. Each link in the diagram represents a contour line and its enclosing contour (the lower of the two). Each line may have many upper neighbors, but may have only one enclosing neighbor. Each branch in the tree represents a divergence, where there are two or more contours of the same level that are enclosed by a common lower neighbor. On the terrain, this is analagous to a pass between two peaks.

Neighbor relationships and the contour tree

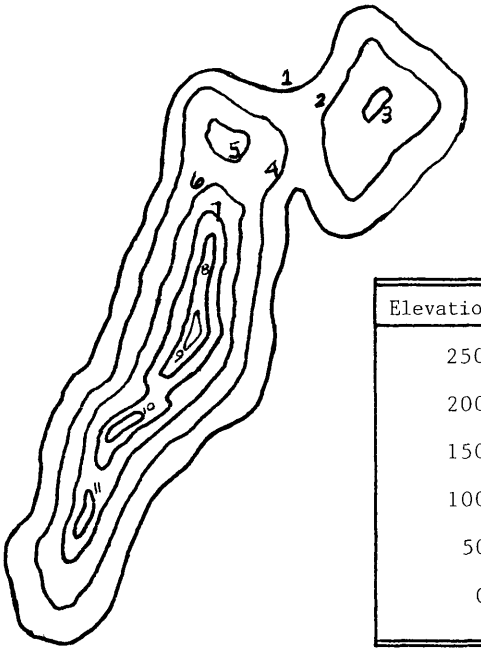
An examination of Table 1 shows that all of the links represent contour lines that are physically close to each other on the contour map. This is reasonable, since those contour lines are also topologically close to each other. For example, the link between line 3 and line 2 in the contour tree indicates that line 3 is nested within line 2, and that these lines are neighbors on the map. This suggests that the ability to recognize neighbor relationships among the contour lines could lead to the deduction of their positions within the contour tree.

Defining neighbors from a scanned contour map image

As a human operator examines a series of line segments that comprise a contour map they are able to recognize the spatial relationships among the lines, and to make assumptions about the topological relationships. For a computer program the process is somewhat different, for although the neighbor relationships can be defined, the topological significance of each line segment cannot be deduced without some additional information. Additionally, while all the topological relationships in the contour tree have their counterparts in the physical relationships, not all of the neighbors on the contour map represent links within the tree structure.

Filtering the neighbor relationships and forming the contour tree

While the definition of the neighbor relationships within a contour map is a primarily mechanical process, the recognition of the significant neighbors (those that fit within the contour tree) is a much tougher conceptual problem.



Contour interval 50 m

Elevation of segment 1 is 0 (Sea level)

Elevations	Level	Tree structure
250	6	9
200	5	8 10
150	4	7 11
100	3	3 5 6
50	2	2 4
0	1	1

Figure 1. Contour map of Strange Island

Table 1. Contour tree for Strange Island

The approach used here makes certain assumptions about 'normal' terrain and about alternative representations of the contour tree. It consists of a series of rules for manipulating the neighbor relationships to create lists of potentially significant neighbors, and a set of rules for the filtering of these potentially significant neighbors for subsequent assignment into the contour tree.

The tasks of the algorithm are:
- Recognition of the neighbors in a scanned contour image,
- Creating lists of potentially significant neighbors,
- Examining these lists and assigning each segment into the contour tree structure.

Each of these will be discussed more fully below.

DEFINING NEIGHBOR RELATIONSHIPS FROM SCANNED IMAGES

Figure 2 shows a portion of a typical contour map. The position of each of the line segments is captured by scanning the image and producing a computer file that represents each small area on the map as either a line or a blank space (Figure 3). The line segments from the original map can then be reconstructed by connecting those pixels that are next to each other, and assigning an arbitrary identifying number to each. The pixels in the scanned image may be thought of as a grid of graph paper, with each square representing one pixel. In Figure 4 each

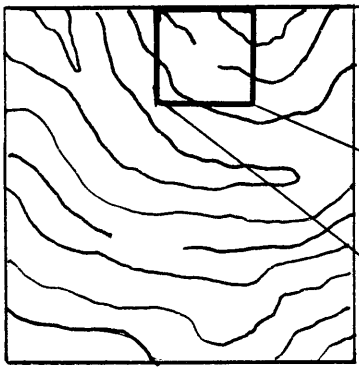


Figure 2. A typical contour map with gaps.

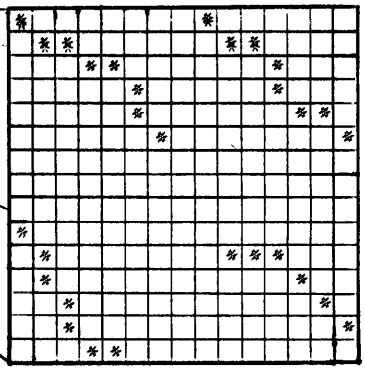


Figure 3. All pixels are 'on' or 'off'.

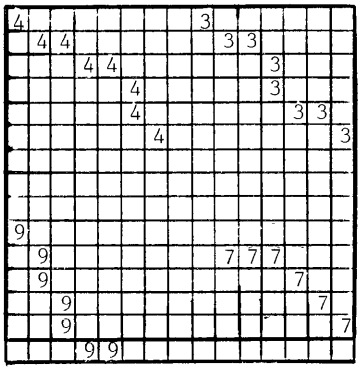


Figure 4. Each 'on' pixel is assigned an identification.

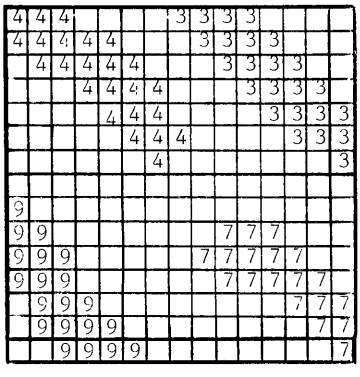


Figure 5. After the first round of propagation.

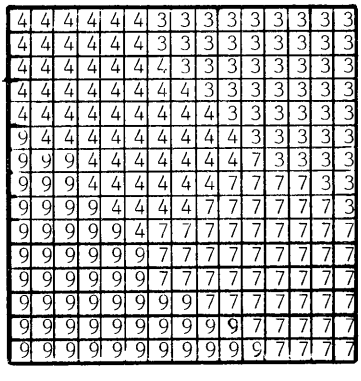


Figure 6. Propagation continues until all space is filled.

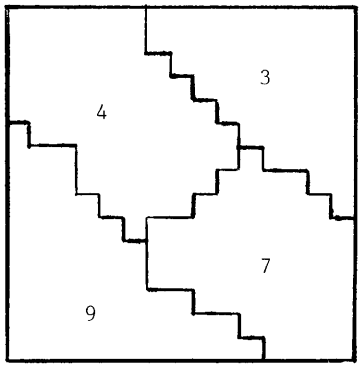


Figure 7. Neighbors and the length of the boundaries can now be determined.

pixel is now assigned a number, rather than being either on or off.

In our example, there are four line segments within the small area, 3, 4, 7, and 9. The method used by the computer program to define the neighbors is to propagate (spread the influence of) each line segment by assigning to each blank space that is adjacent to an assigned pixel the identification number of the assigned pixel (Figure 5). This is done one step at a time until all of the blank space is filled (Figure 6). At this stage it is only necessary to examine the interfaces between the individual line segments and to count the length of each boundary (Figure 7).

After the propagation is complete and the neighbors have been established, the information is compiled into a table that gives the following information for each line segment.

- The number of neighbors
- A list of the neighbors, sorted by length of boundary
- For each neighbor, the absolute length of the boundary

Table 2 is a summary of the neighbor relationship table that corresponds to the Strange Island example. The symbols (+ and -) represent relatively long and short neighbors, compared to the length of the longest neighbor. In practice, these measures are given as a percentage of the longest neighbor, but for the sake of illustration the (+ -) symbols will be sufficient. In general, the long neighbor relationships tend to indicate roughly parallel line segments, while short neighbor relationships tend to indicate knolls, gaps in lines, or other less significant features. An examination of the contour map (Figure 1) and the neighbor relationship table (Table 2) bears out these intuitive guidelines.

CREATING LISTS OF SIGNIFICANT NEIGHBORS

The most significant neighbor relationships are the ones in which one contour is enclosed by its lower neigbor. In an effort to identify these, the program tries to identify local peaks and to create a list of successively lower contours, each one enclosing the line segment above it. These lists are also called traverses.

Local peaks may be identified by the fact that they have only one neighbor, or by operator input. In practice, pits may also appear as one-neighbor segments, so it is important for the operator to identify peaks and pits from the list of one-neighbor segments. Once a traverse has been started, line segments are added to the list by examining the neighbor relationship table and using the rules below. After a segment has been added to the traverse list, its neighbors are considered (excluding the neighbor that already occurs in the tree) for inclusion in the tree. This process is continued until no other segments may be added, then the next traverse is formed.

Table 2. Neighbor relationships and the correct tree structure, by individual line segments

Line segment	Longest neighbor	Other neighbors (by length)	Lower neighbor	Level
1	4	2-	root	1
2	1	3- 4-	1	2
3	2	*	2	3
4	1	6+ 2- 5-	1	2
5	4	6-	4	3
6	4	7+ 11- 5-	4	3
7	6	8- 10-	6	4
8	7	9- 10-	7	5
9	8	*	8	6
10	7	8-	7	5
11	6	*	6	4

* indicates one-neighbor segments
\+ indicates a relatively long neighbor
\- indicates a relatively short neighbor

Table 3. Traverses derived from the neighbor relationship table (Table 2)

One-neighbor segments				Operator-identified segments			
Col 1	2	3	4	5	6	7	
3	9	11	11	10	5	5	
2	8	6	6	7	4	4	
1	7	4	7	6	1	6	
	6	1		4		7	
	4			1			
	1						

Rules for forming traverses

Rule 1. Start a traverse at a local peak. Use rules 2-5 to add as many elements as possible.
Rule 2. The traverse is ended if there are no neighbors to consider, or if any entry in the list is a pit.
Rule 3. If there is only one neighbor to consider, that neighbor is added to the traverse, and rules 2-5 are invoked again.
If there are two or more neighbors to consider:
Rule 4. If one of the remaining neighbors is significantly longer than the others, this neighbor is added to the list, and rules 2-5 are invoked again.
Rule 5. If the two longest remaining neighbors are about the same length, the traverse is split into two traverses, one for each of the candidates, and rules 2-4 are invoked.

Using the rules to form traverses (an example)

The use of these rules will be demonstrated with the Strange Island example (Table 2). Examination of the table shows that there are three segments that have only one neighbor (3, 9, 11). These traverses are generated in an arbitrary order, generally in ascending order.

Column 1 of Table 3 shows the traverse generated from segment 3. Segment 3 begins the traverse according to Rule 1. Segment 3 has only one neighbor (segment 2), so segment 2 is added to the list according to Rule 3. Segment 2 has three neighbors, but one of them is segment 3, which has already been added to the list, so that one is not considered. That leaves two neighbors to consider (1 and 4). Since segment 1 is significantly longer than segment 4, segment 1 is added to the list according to Rule 4. Since segment 1 is a pit the traverse is ended according to rule 2.

This explanation may be abbreviated in the following form:

Segment added to traverse (start at peak)	Neighbors to consider (excluding previous traverse element)	Selection	Rule used
3	2	2	3
2	1+ 4-	1	4
1	4	end	2

In a similar manner, the traverse in column 2 is shown:

Segment added	Neighbors to consider	Selection	Rule
9	8	8	3
8	7+ 10-	7	4
7	6+ 11-	6	4
6	4+ 11- 5-	4	4
4	1+ 2- 5-	1	4
1	2	end	2

Columns 3 and 4 are generated from segment 11:

Segment added	Neighbors to consider	Selection	Rule
11	6	6	3
6	4+ 7+ 5-	4,7	5

Rule 5 creates two new traverses.

Column 3

11			
6			
4	1+ 2- 5-	1	4
1	2	end	2

Column 4

11			
6			
7	8- 10-	end	none

In the previous examples, all of the traverses began with one-neighbor segments. While the program can generate some correct traverses with almost no operator assistance, the ability of the program to recognize peaks and pits (even if they are not one-neighbor segments) improves the performance of the program quite significantly and involves very little additional operator intervention.

Columns 5, 6, and 7 represent traverses that are possible if segments 5 and 10 are recognized as local peaks through operator input. They are generated in the same manner as the previous examples.

Segment added	Neighbors to consider	Selection	Rule
10	7+ 8-	7	4
7	6+ 8-	6	4
6	4+ 11- 5-	4	4
4	1+ 2- 5-	1	4
1	2	end	2

5	4+ 6-	4	4
4	1+ 6+ 2-	1,6	5

Rule 5 starts two new traverses

Column 6

5			
4			
1	2	end	2

Column 7

5			
4			
6	7+ 11- 5-	7	4
7	8- 10-	end	none

With the addition of columns 5-7, all of the line segments from the original map are accounted for and may be assigned into the tree structure. In the absence of the last three columns, segments 5 and 10 would have been unaccounted for.

Up to this point, several relationships have been isolated as potentially significant, but not all of these are valid. The next step in the solution is to manipulate these traverses with the aim of extracting the contour tree structure from them.

ASSIGNING LINE SEGMENTS INTO CONTOUR TREE POSITIONS

The position of a line segment within the contour tree is defined by the level of the contour and its lower neighbor, since each element may have only one lower neighbor. Each level in the contour tree represents an elevation difference of one contour interval, with the root having the lowest elevation (Figure 1). The program must examine the traverses and determine the lower neighbor of each line segment, and the level of that segment relative to the lowest pit.

If a traverse that runs from a peak to a pit is defined correctly, it can be assigned positions in the tree, working from the pit and proceeding 'uphill'. As an example, consider the traverse 9-8-7-6-4-1. Starting with segment 1 and going towards segment 9, each element in the list is the lower neighbor of the following segment, and each element is one level higher than its lower neighbor. These can be seen to be the correct relationships in the contour tree (Table 1). If a traverse contains any contradictions in either level or lower neighbor, the program must detect it and ignore it. The following guidelines are followed when assigning traverse elements into the tree structure:

* Reverse the order of all traverses, so that the pits are first and the peaks are last. This facilitates going uphill.

* The correct choice of the first traverse to be processed is important, as it may influence the outcome of the tree. The best choice appears to be to select the longest traverse that begins with the lowest pit and ends in a peak (column 2 of Table 3).

* The first element is added arbitrarily (One has to start somewhere). Thereafter, each element is checked to see if it has already been assigned into the tree. If it is not in the tree already, it is added. The previous element is the segments lower neighbor, and the segments level is one higher than its lower neighbor. If it already exists in the tree, then its level and lower neighbor assignments are checked with the values it would have been assigned. If there are any contradictions, the remainder of the traverse is disregarded. As long as elements are either added to the tree or confirmed in their position, each subsequent element of a traverse may be considered for addition into the tree.

Consider the traverses of the Strange Island example. Start with the longest traverse that qualifies (column 2). Segment 1 is the lowest pit and is assigned to be level 1, with no lower neighbor (it is the root). As each element in the list is considered, it will be found not to be in the tree, so each will be assigned the next higher level, and the appropriate lower neighbor (Table 2).

Generally, the rest of the traverses are processed according to the number of elements (longer ones going first). The next traverse to consider is column 3 (ignoring columns 5-7 for now). Segment 1 exists in the tree, so the program assigns the current level and lower neighbor to match the values alreay assigned. Now segment 4 is considered. It also exists in the tree, so its assigned level and lower neighbor are checked with the expected values (segment 1 as lower neighbor and level 2). These values match, so segment 6 is checked, and it matches also, so the current levels are updated. Now, when segment 11 is considered, it is not found in the tree, so it is added, above segment 6, in its proper place in the tree.

When column 4 of Table 3 is considered, however, some contradictions occur. The first element (segment 7) occurs in the tree, so it is assumed to be correct. Now, when segment 6 is considered, it is also in the tree, so its consistency is examined. If it were correct in this traverse, segment 6 would be one level higher than segment 7. The opposite is true, however, (segment 6 is lower than segment 7 in the original traverse) so the remainder of this traverse is rejected. This is just as it should be, since this traverse represents incorrect relationships. It can be seen now why it is so important to choose an initial traverse that has correct relationships, because if the first traverse has inconsistencies, those will be regarded as the correct relationship.

By processing the traverses, the contour tree structure is recovered from the neighbor relationships, and most of the contours can be assigned their correct elevation. Furthermore, the line segments that the program is unable to assign are pointed out to the operator for manual input.

Operator intervention can be minimized, usually confined to delineating pits and peaks, assigning an elevation to at least one contour (preferrably the lowest), and giving the contour interval. Additionally, at the conclusion of the program the operator can confirm or correct information provided by the program, and can handle the situations that the program is unable to untangle.

In practice, the contour line separation plates for topographic maps are the most common source for contour information. These printing plates routinely have gaps for labelling and other interruptions, and may contain other inconsistencies. The program was designed to consider incomplete data, and research was carried out to determine the programs performance. A small area containing 55 line segments was tested with various degrees of errors introduced. The initial research included very little operator assistance (only the lowest contour and the contour interval) and produced fair results. The program was able to assign more than 80 percent of the lines in the complete contour map, with graceful degredation as increasing percentages of gaps were introduced. With gaps in 20% of the line segments, the program began to fail in the identification of the surface structure.

However, most of these problems are avoided with the minimal operator input described earlier. The program can then be expected to annotate almost all of the segments, thus having great potential for significantly reducing the time and effort of extracting heights from scanned contour maps.

REFERENCES

Mark, D.M. 1977, Topological Randomness of Geomorphic Surfaces. PhD dissertation, Simon Fraser University.

Morse, S.P. 1965, A Mathematical Model for the Analysis of Contour Line Data. Tech Rep 400-124, New York University.

Morse, S.P., 1969, Concepts of Use in Computer Map Processing. Communications, Assoc. Comp. Mach. v12, pp 145-152.