Sequential Labeling

Iterative Algorithm (Haralick 1981)

- no auxiliary storage to produce the labeld image from the binary image.
- useful in environments whose storage is severely limited.

1. initialization step
2. repeat

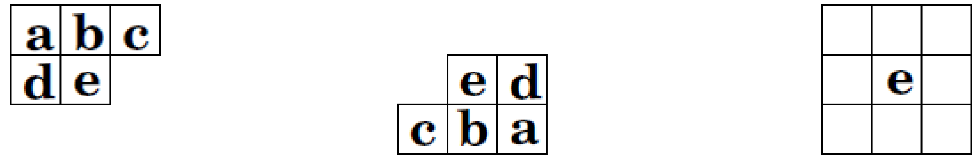
    top-down & left-right label propagation

    bottom-up & right-left label propagation

   until no changes occur

```
procedure Iterate;
"Initialization of each 1-pixel to a unique label"
for L:=1 to NLINES do
        for P:=1 to NCOLUMNS do
                if  I(L,P) =1
                then LABEL(L,P):=NEWLABEL()
                else LABEL(L,P):=0
        end for
end for;
```

Sequential Labeling



"Top-down passes; Bottom up passes; 8-connected neighborhood"

"**procedure** Iterate – page 2"

"Iteration of top-down followed by bottom-up passes"

```
repeat
"Top-down passes"
CHANGE:=false;
 for L:=1 to NLINES do
          for P:=1 to NCOLUMNS do
                    if  LABEL(L,P)<>0 then
                              begin
                                        M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                        if M<> LABEL(L,P)
                                        then CHANGE:=true;
                                        LABEL(L,P):=M
                              end
          end for
 end for;
```

Sequential Labeling

“**procedure** Iterate – page 3”

“Bottom-up pass”

```
for L:= NLINES to 1 by –1 do
        for P:= NCOLUMNS to 1 by –1 do
                if  LABEL(L,P)<>0 then
                        begin
                                M:=MIN(LABELS(NEIGHBORS(L,P)U(L,P)));
                                if M<> LABEL(L,P)
                                then CHANGE:=true;
                                LABEL(L,P):=M
                        end
        end for
end for;

until CHANGE:=false

end Iterate
```


Technical University of Cluj Napoca
Computer Science Department

Sequential Labeling

Example (N4)

	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

1. Initial image

	1	2		3	4	
	5	6		7	8	
	9	10	11	12	13	

2. Initialization

	1	1		3	3	
	1	1		3	3	
	1	1	1	1	1	

3. Top-down & left-right label propagation

	1	1		1	1	
	1	1		1	1	
	1	1	1	1	1	

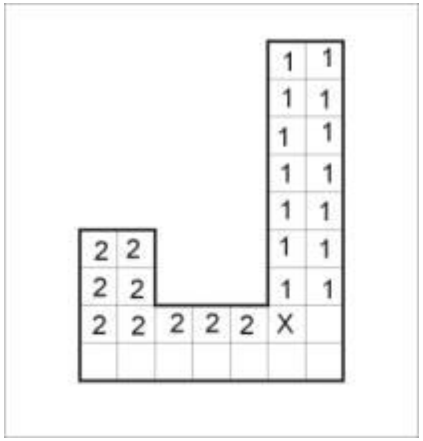
4. Bottom-up & right-left label propagation

Technical University of Cluj Napoca
Computer Science Department

Classical Algorithm

- Based on the classical connected components algorithm for graphs.
- 2 passes through the image but requires a large global table for recording equivalences.

1. First pass: performs label propagation, much as described above.
   - Whenever a situation arises in which two different labels can propagate to the same pixel, the smaller label propagates and each such equivalence found is entered in an equivalence table (e.g. (1,2) $\rightarrow$ EqTable).
   - Each entry in the equivalence table consists of an ordered pair, the values of its components being the labels found to be equivalent.
   - After the first pass, the equivalence classes are found.
   - Each equivalence class is assigned a unique label, usually the minimum (or oldest) label in the class.



2. A second pass through the image performs a translation, assigning to each pixel the label of the equivalence class of its 1-st pass label.

Classical Algorithm

```
procedure Classical
"Initialize global equivalence table"
EQTABLE:=CREATE();

"Top-down pass 1"
for L:= 1 to NLINES do
        "Initialize all labels on line L to zero"
        for P:= 1 to NCOLUMNS do
                LABEL(L,P):=0
        end for
        "Process the line"
        for P:=1 to NCOLUMNS do
                if I(L,P):= 1 then
                        begin
                                A:= NEIGHBORS((L,P));
                                if ISEMPTY(A)
                                then M:=NEWLABEL()
                                else M:= MIN(LABELS(A));
                                LABEL(L,P):=M;
                                for X in LABELS(A) and X<>M
                                        ADD(X, M, EQTABLE)
                                end for;
                        end
        end for
end for;
```


Technical University of Cluj Napoca

Computer Science Department

Classical Algorithm

```
"Find the equivalence classes"
EQCLASSES:=Resolve(EQTABLE);

"Find the equivalence label of an equivalence class"
for E in EQCLASSES
        EQLABEL(E):= min(LABELS(E))
end for;

"Top-down pass 2"
for L:= 1 to NLINES do
        for P:= 1 to NCOLUMNS do
                if I(L,P) = 1
                then LABEL(L,P):=EQLABEL(CLASS(LABEL(L,P)))
        end for
end for
end Classical
```

- RESOLVE - algorithm for finding the connected components of the graph structure, defined by the set of equivalences (EQTABLE) defined in pass 1.
- The main problem with the classical algorithm is the global equivalence table (large images with many regions, the equivalence table can become very large)

Technical University of Cluj Napoca
Computer Science Department

Classical Algorithm

Example (N4)

1					1	1
		1	1			1
		1				1
		1				1
1	1	1				1
	1	1		1		1
	1	1	1	1		1
				1	1	1

1. Initial image

1					2	2
		3	3			2
		3				2
		3				2
4	4	3				2
	4	3		5		2
	4	3	3	3		2
				3	3	2

2. Top down (pass 1)

EQTABLE:
(4, 3), (3, 5), (3, 2) …

EQCLASSES:
1: {4, 3, 5, 2}
2: (6,8,9, …}
….
n: {….}

EQLABEL:
1, 2
2, 6
…
n, x

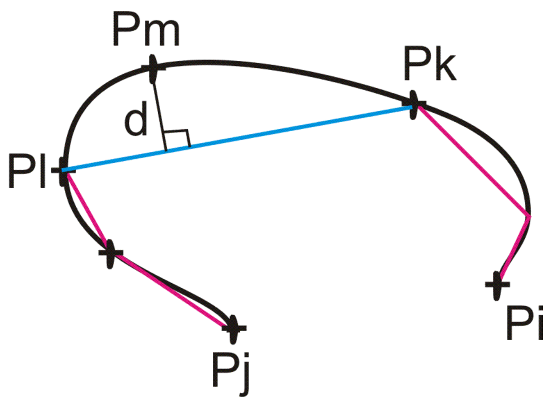
Polygonal Approximation

Polygonal approximation of contours

Curve C: $f(x,y)=0 \Rightarrow$ polygon that closely approximates C with an error smaller than ε and having a number of vertices as small as possible**:**

- Any polygonal fitting algorithm requires that the data points be subdivided into groups, each one of them to be approximated by a side of the polygon.
- The first simplification of the polygon fit problem is to draw a line between the endpoints of each group rather than search for the optimal solution.
- If the approximation error is too big the group could be split in two and so one until the error becomes acceptable.
- Let Q a contour consisting of P_i (x_i,y_i) where i=1,2,…,n, and ε the error threshold.

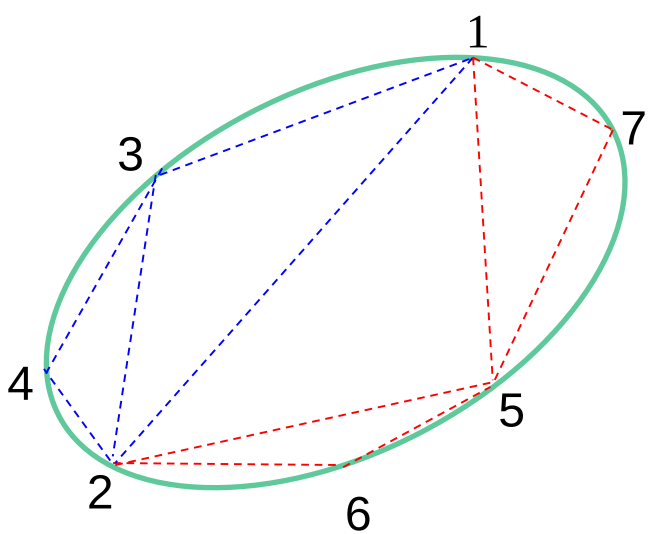
Polygonal Approximation



Procedure POLIGONAL_APROX(Q)
begin
A:=Create_List();
B:=Create_List();
i=Index_of_first_point(Q);
j=Index_of_the_most_far_point(Q);
Insert(j,A); Insert(j,B);
Insert(i,A);
while((A!=NULL)
{
Let k and l the indexes of the last elements of the lists A and B;
Let P_k P_l the segment generated by these two points;
Let m the index of the most far point to P_kP_l segment among the contour points starting with P_k and ending with P_l, when the contour is scanned in counterclock-wise direction.
if ((d=Distance($P_kP_{l,}$, P_m) > ε)
then Insert(m, A)
else { Delete(k, A)
Insert(k, B); }
}
end

$$\text{distance}(P_1, P_2, (x_0, y_0)) = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}.$$

Polygonal Approximation – Example



A	B
2 1 3 4	2
2 1 3	2 4
2 1	2 4 3
2	2 4 3 1
2 5	2 4 3 1

A	B
2 5 7	2 4 3 1
2 5	2 4 3 1 7
2	2 4 3 1 7 5
2 6	2 4 3 1 7 5

A	B
2	2 4 3 1 7 5 6
	2 4 3 1 7 5 6 2

Technical University of Cluj Napoca
Computer Science Department