A Simple and Efficient Connected Components Labeling Algorithm

Luigi Di Stefano
DEIS, University of Bologna
Via Risorgimento 2, 40136 Bologna, Italy
ldistefano@deis.unibo.it

Andrea Bulgarelli
University of Modena
Via Campi 213/b, 41100 Modena, Italy
andbulga@risorsei.it

Abstract

We describe a two-scan algorithm for labeling connected components in binary images in raster format. Unlike the classical two-scan approach, our algorithm processes equivalences during the first scan by merging equivalence classes as soon as a new equivalence is found. We show that this significantly improves the efficiency of the labeling process with respect to the classical approach. The data-structure used to support the handling of equivalences is a 1D-array. This renders the more frequent operation of finding class identifiers very fast, while the less-frequent class-merging operation has a relatively high computational cost. Nonetheless, it is possible to reduce significantly the mergings cost by two slight modifications to algorithm's basic structure. The ideas of merging equivalence classes is present also in Samet's general labeling algorithm. However, when considering the case of binary images in raster format this algorithm is much more complex than the one we describe in this paper.

1. Introduction

In binary images analysis objects are usually extracted by means of the connected components labeling operation, which consists in assigning a unique label to each maximal connected region of foreground pixels. The classical sequential labeling algorithm dates back to the early days of computer vision [1], [2] and relies on two subsequent raster-scans of the image. In the first a temporary label is assigned to each foreground pixel based on the values of its neighbours already visited by the scan. When a foreground pixel with two foreground neighbours carrying different labels is found, the labels associated with the pixels in the neighbourhood are registered as being equivalent. Throughout the paper we will refer to this situation also as to a "conflict". After completion of the first scan equivalences are processed to determine equivalence classes. Then, a second scan is run over the image so as to replace each temporary label by the identifier of its corresponding equivalence class.

In this paper we describe a two-scan labeling algorithm whereby, unlike the classical approach, equivalences are processed during the first pass in order to determine the correct state of equivalence classes at each time of the scan. This is obtained by merging classes as soon as a new equivalence is found, the data structure used to support the merging being a simple 1D array. This approach allows the check for a conflict to be carried out on class identifiers rather than on labels, as instead it is mandatory with the classical algorithm. We show that this significantly improves the efficiency of the labeling process.

The ideas of merging equivalence classes and performing conflict checks class identifiers is present also in the algorithm by Samet et. al. [7], [8]. However, this algorithm is much more complex than the one we describe in this paper. In addition, the authors do not point out that performing conflict checks directly on equivalence classes improves the efficiency of the labeling process.

2. The connected components labeling problem and the classical approach

Let I be a binary image and F, B the subsets of I corresponding respectively to foreground and background pixels. A connected component of I, here referred to as C, is a subset of F of maximal size such that all the pixels in C are connected.

Two pixels, P and Q, are connected if there exists a path of pixels $(p_0, p_1 \ldots p_n)$ such that $p_o = P$, $p_n = Q$ and $\forall\ 1 \leq i \leq n$, p_{i-1} and p_i are neighbours.

Hence, the definition of connected component relies on that of a pixel's neighbourhood: if this includes 4-neighbours C is said to be 4-connected otherwise if we assume that a pixel has 8-neighbours, C is said to be 8-connected.

Given I, the connected components labeling problem consists in generating a new image in which a unique label

is assigned to pixels belonging to the same connected component of I and different labels are associated with distinct components.

As for background pixels, they are typically left unchanged by the labeling process, though in principle it is possible to mark them with a new label value. Hence, assuming that I has n connected components, in the labeled image we find $n+1$ labels, one of them, l_B, assigned to the pixels in B and the remaining n used to mark the pixels in F as belonging to a distinct component. It is worth noticing that the relation among pixels expressed through a label value in the labeled image depends on the label value: two pixels marked with l_B belong to B, but they are not necessarily connected, two pixels marked with $l_i \neq l_B$ belong to F and are connected.

The classical sequential algorithm for labeling connected components consists of two subsequent raster-scans of I [1], [2]. In the first scan a temporary label is assigned to each pixel in F based on the values of its neighbours already visited by the scan. Indicating as x the pixel to be labeled, the set of the already labeled neighbours, N, is given by $\{p, q, r, s\}$ in case of 8-connectivity and $\{p, q\}$ in case of 4-connectivity:

$$\begin{array}{ccc} p & q & r \\ s & x & \end{array} \qquad \begin{array}{cc} & p \\ q & x \end{array}$$

Calling N_F the set of the already visited neighbours belonging to foreground ($N_F = N \cap F$), the first scan can be described as follows. If N_F is empty x is assigned a new label (x is the first pixel of a new component), if all the pixels in N_F have label l x is assigned l (x belongs to the component already labeled as l), if two pixels in N_F have different labels then x is assigned either of them and the labels associated with the pixels in N_F are registered as being equivalent (x is the point where two previously disjoint components merge together).

As a result of the scan, no temporary label is assigned to pixels belonging to different components but different labels may be associated with the same component. Therefore, after completion of the first scan equivalent labels are sorted into equivalence classes and a unique class identifier is assigned to each class. Then, a second scan is run over the image so as to replace each temporary label by the class identifier of its equivalence class.

Based on this approach, several different ways of handling equivalences have been proposed in the literature. They differ in the data structures used to register equivalences during the first scan as well as in the method employed to obtain equivalence classes between the two scans. We report here a few examples.

In the original paper by Rosenfeld and Pfaltz [1] equivalent labels are stored as $n-$tuples of the form $(l_{i_1} \dots l_{i_n})$ ($n = 2$ for $4-$connectivity, $n = 4$ for $8-$connectivity) into an Equivalence Table (T). After the first scan the entries of T are ordered (in order of increasing first label) and then T is processed so as to build a second table ($T^{'}$) made out of $n-$tuples in which the first label is the smallest representative of an equivalence class and the remaining labels are elements of the class. $T^{'}$ is obtained by scanning T, moving the current entry $(l_{i_1} \dots l_{i_n})$ from T to $T^{'}$ and replacing $l_{i_2} \dots l_{i_n}$ with l_{i_1} in the remaining entries of T.

Haralick and Shapiro [3] point out that the Equivalence Table embodies a graph structure where the nodes are the labels found in the first scan and the edges connect pairs of equivalent labels. Therefore, equivalence classes can be seen as the connected components of the graph structure associated with the Equivalence Table, and these can be determined by means of a standard depth-first search algorithm.

Gonzales and Woods in [4] suggest managing equivalences between labels by means of general, formal tools for handling equivalence relations. In such a framework, equivalent labels pairs may be seen as members of an equivalence relation represented through a boolean $n \times n$ matrix, B (n is the number of labels, $B(i, j) = 1$ means that labels i and j are equivalent). Thus, B can be filled up during the first scan and then the matrix representing the transitive closure of the relation, B^{+}, can be obtained by means of tools such as Warshall's algorithm [5]. A further simple processing step is required to determine equivalence classes from B^{+}.

In the algorithm described by Klette and Zamperoni [6] all the equivalent label pairs are stored as $2-$tuples, $(g_1(i), g_2(i))$, into an Equivalence Table, with $g_2(i) < g_1(i)$ being the temporary label assigned to the pixel under examination. The Equivalence Table is then processed by scanning g_1 and for each $g_1(i)$, scanning g_2 so that if $g_2(k)$ equals $g_1(i)$ then $g_2(k)$ is replaced with $g_2(i)$. The scanning of g_1 is iterated until no further change occurs.

In Figure 1 we show a C description of the classical algorithm in the case of 4-connectivity. Since different data-structures and processing method can be used to store and sort out equivalences, we leave unspecified the details concerning the actual handling of equivalences.

3. Basic structure of the proposed algorithm

In our labeling algorithm equivalences are processed directly in the first scan so that equivalence classes are always maintained updated during the scan. This is obtained by associating a new equivalence class with each new label and by merging the corresponding classes as soon as a new equivalence is found.

We perform the merging operation using a very simple data structure called *Class Array* (C). C is a one-dimensional array as large as the maximum label value and containing for each label value its corresponding class iden-

```
// lp,lq,lx: labels assigned to p,q,x
// B:background, F:foreground.
// FIRST SCAN:
for(i=1; i<NROWS-1; i++)
for(j=1; j<NCOLS-1; j++) {
if (I[i,j]==F) {
lp = I[i-1,j];
lq = I[i,j-1];
if(lp == B && lq == B) {
NewLabel++;
lx = NewLabel;}
else if((lp != lq)&&(lp != B)&&(lq != B)){
// REGISTER EQUIVALENCE (lp,lq)
lx = lq;}
else if(lq != B) lx = lq;
else if(lp != B) lx = lp;
I[i,j] = lx;} }
// FIND EQUIVALENCE CLASSES
// SECOND SCAN
```

Figure 1. The classical algorithm.

tifier (i.e. $C[i]$ is the equivalence class associated with label i). C is initialised by posing $C[i] = i,\ i = 0 \dots maxlabel$; this means that, initially, each possible label is assumed to belong to a distinct class. Moreover, it is useful to reserve one entry in C to express in the class identifiers domain the relation among pixels implied by the label value used to mark foreground pixels. This is done by setting $C[B] = l_B$, i.e. typically $C[B] = B$. When two labels, l_i and l_j, are found to be equivalent during the first scan, the corresponding classes, $C[l_i]$ and $C[l_j]$, are merged. The merging consists in first setting one of two class identifiers to be the survivor and the other to be deleted, and then storing the survivor identifier into the entries in the Class Array equal to the deleted one. The choice of the survivor can be arbitrary (for example: always retain $C[l_i]$, always retain $C[l_j]$, retain $min(C[l_i], C[l_j]) \dots$). Suppose for example that we always take $C[l_i]$ as the survivor, the C code for the merging step is given by:

```
for(k=0; k<=Newlabel; k++)
if (C[k] == C[lj]) C[k]=C[li];
```

Keeping equivalence classes in their correct, updated state during the first scan allows the check for a new equivalence to be carried out in the class domain rather than in the label domain. More precisely, when a conflict between l_i and l_j is found, we check whether $C[l_i]$ and $C[l_j]$ are different or not and then handle the equivalence (i.e. merge the two classes) only in the former case.

After completion of the first scan the Class Array holds the class identifier associated with each temporary label and thus can be used as a look-up table in the second scan to change label values into their corresponding class identifiers. Figure 2 shows the C code for the proposed algorithm in the 4-connectivity case. As far as examples and experimental results are concerned, we will consider 4-connectivity throughout the paper.

```
// C has been initialised ( C[i]=i );
// FIRST SCAN
for(i=1; i<NROWS-1; i++)
for(j=1; j<NCOLS-1; j++){
if (I[i,j] == F){
lp = I[i-1,j];
lq = I[i,j-1];
if(lp == B && lq == B){
NewLabel++;
lx = NewLabel;}
else if((C[lp]!=C[lq])&&(lp != B)&&(lq != B)){
for(k=0; k<=Newlabel; k++)
if (C[k] == C[lp]) C[k]=C[lq];
lx = lq;}
else if(lq != B) lx = lq;
else if(lp != B) lx = lp;
I[i,j] = lx;}}
// SECOND SCAN
for(i=1; i<NROWS-1; i++)
for(j=1; j<NCOLS-1; j++)
if (I[i,j] != B) I[i,j]=C[I[i,j]];
```

Figure 2. The proposed algorithm.

The basic advantage associated with performing the checks for a new equivalence in the class domain is as follows. The merging operation grants automatic note making of all the equivalences between the merged classes members which are implied through the transitive property. Hence, performing the check for a new equivalence directly on classes allows the transitive property to be exploited during the first pass: a new equivalence implied through the transitive property by those already found will not be handled by the algorithm (i.e will not cause a merging operation).

On the other hand, with the classical algorithm equivalence classes are sorted out only after the first scan and therefore, during the first scan, it is mandatory to perform the check for a new equivalence in the label domain. This means that all the equivalences found during the scan, including those implied by previous ones, must be handled by the algorithm (i.e must be registered and then processed).

Whatever the actual method used to manage equivalences, it is clear that handling a new equivalence results in a computational cost for the labeling process. Therefore, the handling of useless equivalences can be seen as a computational overhead. By useless equivalence we mean an equivalence which does not carry any new information on equivalence classes. In this context, unlike the classical approach, the proposed algorithm is intrinsically efficient

since, given the image and the scan order, it does not handle useless equivalences. In the next section we will address this issue in more details.

The ideas of merging equivalence classes and performing the check for conflicts on class identifiers is present also in the algorithm by Samet et. al. [7], [8]. They propose a general labeling algorithm capable of handling in a unified way a wide range of image representation schemes (such as two-dimensional arrays, run lengths, bintrees, quadtrees ...) and making use of a tree structure based on father links to represent each equivalence class. However, when considering the 2D-array image representation, Samet's algorithm turns out to be significantly more complex than the one proposed in this paper. In addition, the authors do not point out the better efficiency associated with performing the conflict checks in the class domain with respect to the classical approach. Finally, although the tree structure is very efficient for merging classes when a new conflict is found, it requires traversing the tree from the node representing the label up to the root when a class identifier is to be found for the purpose of checking label conflicts. Instead, our 1D-array renders the check for a conflict faster than the class merging operation. This seems to us more appropriate since the mergings are executed as a result of a -typically small- subset of the conflict check operations.

4. Efficiency of the proposed algorithm

In this section, we show first by means of two examples that, thanks to the merging of classes and to the strategy of performing checks on the basis of class identifiers, the proposed algorithm does not handle useless equivalences that instead would be registered and processed using the classical algorithm. Then, we report some experimental results aimed at assessing the actual impact of this property in practical cases.

Figure 3 (top) shows a simple shape in which our algorithm does not handle a conflict because the labels involved are yet known to be equivalent due to the transitive property. Initially, C is set to $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$; after finding the equivalence $(2,1)$, C is updated to $\begin{bmatrix} 2 & 2 & 3 \end{bmatrix}$ (we assume to retain label l_q and class $C[l_q]$); then, after finding $(3,2)$ C becomes $\begin{bmatrix} 3 & 3 & 3 \end{bmatrix}$. Thus, when we get to the equivalence $(3,1)$, this is not handled by the algorithm since $C[3] = C[1]$. Instead, with the classical algorithm we would register $(3,1)$ during the first pass as well as process it before the second pass.

During the first scan two labels may cause multiple conflicts. In the classical algorithm this situation is typically not addressed, the same equivalence being registered as many times as it is occurs. This choice depends on the computational cost of the search in the structure holding equivalences that would be needed to verify whether the current

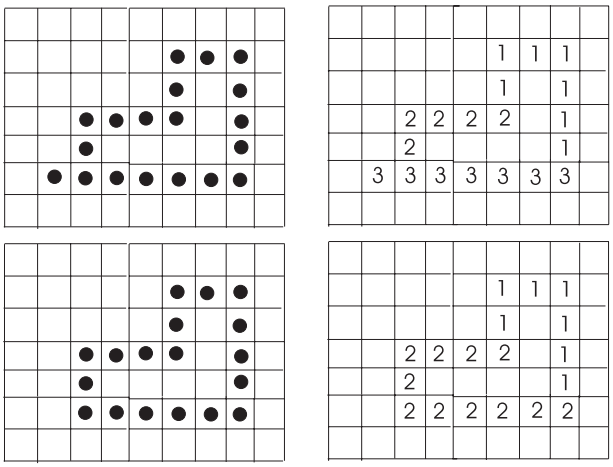

Figure 3. Binary and labeled shapes.

equivalence has been found previously or not. Instead, with our algorithm once the Class Array has been updated according to a given equivalence, any new occurrence of this equivalence will be ignored by the algorithm. Figure 3 (bottom) shows such a situation: after finding $(2,1)$ we set $C[1] = C[2] = 2$ and therefore the second occurrence of $(2,1)$ in the last row is not handled.

We have considered a set of test-images and evaluated the amount of conflicts found by the classical algorithm and by our algorithm. The test-images include binary as well as 8-bit gray-scale images; the latter ones have been binarised using an automatic threshold selection method [9]. Most images have been taken from the Image Library of the HIPR package. [10]. The images can be viewed at http://www-labvisione.deis.unibo.it/ ldistefano/

The results, reported in Table 1, demonstrate that in practical cases the property of ignoring useless equivalences can significantly reduce the amount of conflicts that need to be handled by the labeling process.

5. Reducing the cost of class merging

As already mentioned in Section 3, C is conceived so as to "make the common case fast", i.e. so as to find quickly class identifiers from label values. On the other hand, the less frequent merging operation has a higher computational cost since it requires a complete scan of the Class Array (up to the last used label value, namely *Newlabel*). However, it is possible to reduce the global cost of the mergings by avoiding a number of useless scans. In fact, if either of the two classes involved in the merging contains just a single label we can simply set the entry in C associated with this label to the other class identifier. In such a situation the

Image	Classical Algorithm	Proposed Algorithm
cel4	281	78
fun1	2527	1318
hnd1	742	473
hse4	262	124
mon1	6404	224
pcb1	1025	138
pcb2	2449	260
rods	1158	88
chocs	2390	103
son3	1317	425
tls1	7222	224
txt2	380	198
txt3	168	138
wrm1	1483	221

Table 1. Number of conflicts handled by the labeling process.

merging does not require scanning the Class Array for it can be carried out by means of a single assignment. It is worth observing that a necessary condition for a class $C[l_i]$ to contain just a single label is $C[l_i] = l_i$. In fact, this relation holds from the creation of a new label equal to l_i (and corresponding class $C[l_i]$ containing label l_i only) until the involvement of label-class l_i into a merging with, say, class $C[l_j]$, at which time, should $C[l_j]$ be chosen as the survivor, $C[l_i]$ would be changed into $C[l_j]$ (thus rendering $C[l_i] \neq l_i$).

The necessary condition $C[l_i] = l_i$ can be rendered also sufficient very easily in the case $l_i = Newlabel$. Indeed, introducing the rule that if the label-class identifier *Newlabel* gets involved in a merging then it is always chosen as the one to be deleted, we can ensure that, throughout the first scan, the current last-class created by the labeling process must contain a single label and therefore, should it be involved in a merging, this can be carried out through a single assignment. Figure 4 reports the C code of a modified version of the merging algorithm which exploits the aforementioned rule.

```
if (C[li] == Newlabel) C[li] = C[lj];
else if (C[lj]==Newlabel) C[lj]=C[li];
else
for(k=0; k<=Newlabel; k++)
if (C[k] == C[lj])
C[k]=C[li];
```

Figure 4. Modified merging algorithm.

Figure 5 shows an example in which the modified version of the merging algorithm is particularly effective in avoiding useless scans of the Class Array. With the old merging algorithm we would scan C for each of the mergings caused by the equivalences found along the 45 degree-oriented border of the object; on the other hand, the modified version of the algorithm allows all the mergings to be executed through a single assignement since all of them involve the current last-class.

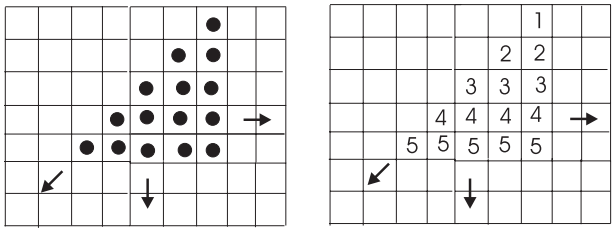

Figure 5. Binary and labeled shape.

Generally speaking, during the first scan we may find many situations in which either of the two classes involved in a merging contains only one label, with this not necessarily being the current last-class. Hence, the number of fast, single-assignement mergings is potentially higher than that attainable by the modified merging algorithm. We can therefore extend the previous approach so as to keep track for each class of whether it is a single-label class or not, and then exploit this information to perform a single-assignement merging when a single-label class is involved in a merging operation. The C code in Figure 6 shows this extended merging algorithm, in which a second one-dimensional array, F, is employed to store the class status information ($F[i] = 1$: Class i contain a single label, $F[i] = 0$: Class i contain more than a label).

```
// F has been initialised (F[i]=1 for all i).
if (F[C[li]] == 1)
{C[li] = C[lj]; F[C[lj]] = 0;}
else if (F[C[lj]] == 1) {
C[lj] = C[li]; F[C[li]] = 0;}
else for(k=0; k<=Newlabel; k++)
 if(C[k]==C[lj]) C[k]=C[li]);
```

Figure 6. Extended merging algorithm.

Eventually, we show the experimental results obtained using the different merging approaches on our set of test-images. Table 2 reports the number of full-scan mergings executed by the labeling process with the basic, modified and extended version of the merging algorithm.

These results show that the extended merging algorithm can dramatically reduce the number of full-scans of the Class-Array which must be executed as a result of a conflict. In turn, also the simpler modified algorithm can yield

a significant reduction of the scans, the two approaches being practically equivalent in a number of cases.

Image	Basic	Modified	Extended
cel4	78	5	1
fun1	1318	249	94
hnd1	473	61	25
hse4	124	19	3
mon1	224	11	9
pcb1	138	18	6
pcb2	260	13	4
rods	88	0	0
chocs	103	3	3
son3	425	223	42
tls1	224	2	2
txt2	198	23	8
txt3	138	10	2
wrm1	221	7	4

Table 2. Comparison between the different merging algorithms.

6. Conclusion

We have described a two-scan labeling algorithm capable of handling equivalences during the first scan by means of a class-merging step. This approach is different from the classical two-scan labeling algorithm and can be found also in the very general labeling algorithm by Samet et.al. The major merits of the proposed algorithm rely its efficiency and simplicity. As regards efficiency, we have shown by means of examples and experimental results that, unlike the classical approach, the proposed algorithm does not handle useless equivalences. As for simplicity, we think that compared to Samet's, our algorithm is remarkably simpler both to understand and to implement. We have provided throughout the paper several C-code descriptions that should effectively support this statement and allow a straightforward implementation of the algorithm.

In the proposed algorithm the data-structure used to support the handling of equivalences naturally favours the more frequent class-finding operation with respect to class-merging. We have shown two simple modifications to the class merging step yielding a significant reduction of the global computational cost associated with the mergings.

To the purpose of focusing on algorithm's fundamentals we have shown code and examples in the 4-connectivity case only. Accordingly, the experimental results are relative to 4-connectivity. Nonetheless, the proposed algorithm can be employed also in the case of 8-connectivity. All the code associated with the class merging operations can be left unchanged while in the code shown in Figure 2 one should simply carry out slightly different checks in order find out whether two temporary labels are equivalent or not.

Acknowledgments

We thank the authors of HIPR and John Wiley and Sons for the use of images *wrm1, son3, txt2, hnd1, tls1*. In addition, we thank Ms. S. Flemming for the use of image *hse4*, R. Aguilar-Chongtay of the Department of Artificial Intelligence, University of Edinburgh the use of image *cel4*, Dr. K. Ritz of Scottish Crop Research Institute, and Drs. C. Glasbey and G. W. Horgan of Biomathematics and Statistics Scotland, University of Edinburgh for the use of image *fun1*, Dr. X. Huang of the Department of Artificial Intelligence, University of Edinburgh for the use of image *txt3*, Mr. A. MacLean of the Department of Artificial Intelligence, University of Edinburgh for the use of images *pcb1, pcb2* .

References

[1] A. Rosenfeld, J. Pfaltz. Sequential Operations in Digital Picture Processing. *Journal of the ACM*, 13(4):471-494, October 1966.

[2] A. Rosenfeld, A. C. Kak.*Digital Picture Processing*. Vol.2: 241-242, Academic Press, 1982.

[3] R. Haralick, L. Shapiro. *Computer and Robot Vision*. Vol.1:33-37, Addison-Wesley, 1992.

[4] R. Gonzales, R. Woods. *Digital Image Processing*. 42-45, Addison-Wesley, 1992.

[5] S. Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11-12, January 1962.

[6] R. Klette, P. Zamperoni.*Handbook of Image Processing Operators*. 314-319, John Wiley & Sons, 1996.

[7] H. Samet,M. Tamminen. An Improved Approach to connected component labeling of images. *Proceedings of CVPR'86*, 312-318, 1986.

[8] M. Dillencourt, H. Samet, M. Tamminen. A general approach to connected component labeling for arbitrary image representations. *Journal of the ACM*, 39(2):253-280, April 1992.

[9] N. Otsu. A Threshold Selection Method for Gray-Level Histograms. *IEEE Trans. on Systems, Man and Cybernetics*, 9:62-66, January 1979.

[10] R. Fisher, S. Perkins, A. Walker, E. Wolfart. *Hypermedia Image Processing Reference*. John Wiley & Sons, 1996. (http:// www.wiley.co.uk/electronic/hipr).