Using Equivalent Pair Information for Image Component Labeling - An Optimal Algorithm

Amelia Fong Lochovsky

Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

Using a fixed size window in a raster scan fashion, image component labeling can take time no better than O(N × M) where N × M is the size of the image. Traditionally equivalent labeled pairs are collected and a second phase to merge them is performed later. There is no known algorithm which merge equivalent pairs in time proportional to the number of equivalent pairs i.e. there is no known algorithm that merge the equivalent pairs in time bounded by O(N × M) as the number of equivalent pairs is O(N × M). An algorithm is presented here which combines labeling and merging which achieves this bound. The information about the equivalent pairs is used in the labeling process. This is asymptotically optimal since labeling takes at least O(N × M).

I Introduction

The tasks in machine vision can be roughly divided into three broad classes based on the data structures they use. Low-level tasks operate on large arrays of pixel values. High-level tasks operate on smaller symbolic data which are descriptions of the image or of the scene under analysis. Tasks which link the low and high levels by taking iconic data as input and produce symbolic structures as output are often considered as intermediate-level tasks. This paper addresses an important intermediate-level task in machine vision: labeling the connected components of a binary image. Connected component labeling consists of assigning labels to pixels in the image such that adjacent pixels are given the same labels. There are various approaches to component labeling. Some require random access to the processed image while some assume run length encoding of the image [5], [6], [7]. Algorithms based on sequential scan of the image are attractive to hardware implementation which is important to meet the speed requirement in real-time applications such as in visual inspection [1], [2], [3], [4]. Other studies and applications of component labelling include [8], [9], [10]. There are also much interests in algorithms for component labeling on parallel architectures [14], [15]. Algorithms based on a raster scan of the image usually make use of previous labeled pixels of a fixed size local window. Due to the fixed window size and the sequential fashion of the labeling process, different branches of the same object may be given different labels, and later found to be connected to each other. These labels are considered to be equivalent and must be combined or merged to correctly represent one single object. This approach can be found in [3], [6], [7].

In most discussion on component labelling, the merging of the equivalent labels are seldom elaborated in detail. In [3], an approach for merging equivalent labels and its hardware implementation was given. The technique is $O(m^2)$, where m is the number of equivalent labeled pairs. A variation on the approach in [3] using technique suggested in [11] would reduce the complexity to O(m log m). By the use of tree structures, the algorithm can be further improved [12]. This is a much studied algorithm and has been proven not to be a linear algorithm. Using existing labeling algorithms, the number of equivalent pair generated is bounded by O(N × M), where N × M is the size of the image. The number of distinct labels is also bounded by O(N × M). Even though the labeling algorithm is O(N × M), there is no known algorithm that merge the equivalent pairs in time linear to the number of equivalent pairs in general [17]. In this paper, an algorithm which uses the equivalent pair information for labeling is presented. The algorithm generates far fewer equivalent pairs that need to be merged. However, in the worst case, it is still O(N × M). By combining the labeling and merging process, it is possible to prove that the algorithm performs with time complexity linear in the size of the image. Since labelling in raster fashion takes O(N × M), the algorithm is

asymptotically optimal.

II Using Equivalent Pair information

For the algorithm, we need an equivalent table E, of size equal or greater than the total number of distinct labels. If this table is implemented in hardware, its size has to be set a-priori. With increase in size and complexity of the input image, this table may need to be quite large, as was also observed in [9]. Hence, techniques for handling this table overflows by reusing label locations as described in [13] should be used. For simplicity, we shall assume in this paper that E is large enough to handle our input images and never overflows. Initially, E[(i)] = i for all i. Each element in the table also has a SIZE, a NEXT, a START, a REPF and a REPB components. They are to be used as described below.

For simplicity, consider 3 x 3 window such as

P_1	P_2	P_3
P_4	P	-
-	-	-

where P is the current pixel to be labeled. Throughout this paper, we are assuming 8-adjacency neighbours. That is, for each pixel P, its four horizontal and vertical neighbours plus its four diagonal neighbours are considered adjacent to P. In our algorithm, each pixel P is labeled with a pair (F(P),M(P)). We shall call F(P) the feature label of P, M(P) the merge label of P. The use of feature and merge labels for partial feature information collection will be discussed later. We shall also discuss the implication of the size of the window on the the performance of the algorithm.

We are assuming a binary image as input. Let p(X) the pixel value of X i.e. p(X) = 1 or 0. The labeling process is based on the labels of the pixels in a local window. The labeling of the first line of the image would depend only on the previously labeled pixels of the same line.

(I) Labeling

(1) If $p(P_4) = 1$ then
F(P) := F(A), M(P) := E[M(P_4)]
else
If $p(P_1) = 1$ then
F(P) := F(B), M(P) := E[M(P_1)]
else
If $p(P_2) = 1$ then
F(P) := F(C), M(P) := E[M(P_2)]
else
If $p(P_3) = 1$ then
F(P) := F(D), M(P) := E[M(P_3)]
else
generate new label x, and set
F(P) := M(P) := x

(2) If E[M(P)] ≠ E[M(P_3)] then
generate the equivalent pair
E[M(P_3)],E[M(P)]

This labeling algorithm is similar to algorithms in [3], [7], except for the use of the table E. Some labeling algorithms label the current pixel using the lowest numbered label amongst labels of its adjacent pixels. However, finding the minimum among a set of labels is harder to implement in hardware than the above fixed selection sequence. In comparison, the component labeling algorithm in [9] is designed mainly for execution in software. It finds the equivalent classes at the end of each line, and uses the minimum label in a given class to relabel the current line. However, the method and complexity for obtaining the equivalent classes are not discussed. In our algorithm, the current line is not relabeled and a bottom-up pass through the entire image is not necessary, as is proposed in [9].

The merge label M(P) is generated with the help of the equivalent table E, which is used to guide both the labeling and the generation of the equivalent pairs.

(II) Equivalent Pairs Processing

The list of pairs generated during labeling of a single line are processed before beginning the next line by executing Algorithm 1 below.

Intuitively, Algorithm 1 processes the equivalent pairs (pairs of labels found to be labeling the same objects) generated so far by updating the equivalent table E, which is used to guide subsequent steps of the labeling process, thus reducing the complexity of the relationship of the labels generated.

Algorithm 1 is based on algorithms for disjoint set union problems [17]. However, enough modifications are necessary, both in the data structure

used and in the details involved, to warrant an explanation and an example. As discussed earlier, a linked list implementation using path compression due to Tarjan [12] can be used which can improve the asymptotic running time of the merging algorithm. However an array is used here to allow fast (constant time) table-lookup which is essential to the labelling process. Also this data structure is amenable to hardware implementation. As will be discussed later in the analysis, the improvement of Algorithm 1 by using linked list implementation will not improve the asymptotic running time of the overall labeling and merging algorithm.

We now describe the data structures used. Here two linked lists are needed. Intuitively, all labels equivalent to, say x, are linked together through the NEXT field, where START[x] is the beginning of that list. To process equivalent pair (x, y), the shorter list, say the one with x on it, (this is determined by comparing SIZE[x] and SIZE[y]) is appended to the longer list, the one that includes y. All E[z] for z on the list equivalent to x will be set to E[y]. After Algorithm 1 is executed for this line, z will never appear as labels in the labeling algorithm, only E[z]. Lets call E[z] the representative element of this set of equivalent labels. The labelling algorithm is such that for any pixels to be labelled with any of the label equivalent to the representative, the latter will be used (as the merge label). All current representative labels are kept in a list of representatives which is doubly linked so that insertion and deletions can be performed in constant time. We shall refer to this list as the r-list. REPF and REPB are used to implement the forward and backward links of the list respectively. Headrl indicates the head of the r-list.

Any label in a set of its own is its own representative. However, we do not add this singleton representatives onto the r-list.

At the beginning of each line, when each label is in its own set, processing (x',y') involves changing E[x'] to y'. Hence y' is a representative element and must be added to the r-list. For each subsequent pair (x',y'), without loss of generality, assume SIZE [x'] <= SIZE [y'], If SIZE [x'] = 1 and SIZE [y'] =1, then y' is to be added to the r-list. If SIZE[y'] $\neq$ 1, then y' must have been added onto the r-list already. If SIZE[x'] $\neq$ 1, then SIZE[y'] > 1 as SIZE[x'] $\leq$ SIZE [y']. Then both x' and y' are already on the r-list. We must now delete x' from the r-list as only y' is the representative element after the pair (x',y') is processed. Since the r-list is in random order, we can insert at the head of the list. Deletions can be from anywhere in the list; and can be accomplished in constant time. If there is at least one equivalent pair generated on this line, the r-list will never become empty as there is at least one representative element.

At the end of each line, the r-list is used to reset SIZE and START such that for all representative elements i on the r-list, SIZE[i] is reset to 1 and START[i] is reset to i. This is because only representative elements will appear in the next line as part of an equivalent pair. Hence we can start merging them as if each is in a set of its own and each set is of size = 1. This idea greatly reduces the complexities of the equivalent pairs allowed to be generated, as we shall see in the analysis of the algorithm. The steps of the algorithm is given below.

Initially, headrl=0, SIZE[i]=1 for all i, START[i]=i for all i. NEXT, REPF and REPB are undefined and will be set by Algorithm 1.

Algorithm 1:

(I) For the first equivalent pair generated for this line, let E[x] = x' E[y] = y'

headrl := y'

REPF[y'] := REPB[y'] := 0

E[x] := y'

START[y'] := START[x']

SIZE[y'] := 2

NEXT[x'] := y'

(II) For each subsequent equivalent pair (x,y) generated do the following:

Let E[x] = x'and E[y] = y'

(1) if x' = y' then do nothing

(2) if x' $\neq$ y' then
if SIZE[x'] $\leq$ SIZE[y'] then do
(i) temp := START[y']
(ii) START[y'] := START[x']
(iii) if SIZE[x'] = 1 then do
(a) E[x'] := y'
(b) NEXT[x'] := y'
(c) if SIZE[y'] = 1 then
begin
{ add y' to beginning
of r-list }
REPF[y'] := headrl
REPB[y'] := 0

```
            REPB[headrl] := y'
            headrl := y'
            NEXT[y'] := 0
        end
    else do
     (a) j := START[x']
     (b) E[j] := y'
     (c) while (NEXT[j] ≠ 0) do
         begin
            j := NEXT[j]
            E[j] := y'
         end
     (d) NEXT[j] := temp
     (e) { delete x' from r-list }
         t1 := REPF[x']
         REPF[REPB[x']] := t1
         if t1 ≠ 0 then
              REPB[t1] := REPB[x']

(iv) SIZE[y'] := SIZE[x'] + SIZE[y']
```

if SIZE[y'] < SIZE[x'] then
reverse x',and y' in the above (i) to (iv) steps

(III) At the end of processing all equivalent pairs generated on each line, reset SIZE and START using the r-list as follows.

if headrl ≠ 0 then do

```
(1) j:= headrl
(2) SIZE[j] := 1
(3) START[j] := j
(4) while REPF[j] ≠ 0 then
        begin
              j := REPF[j]
              SIZE[j] := 1
              START[j] := j
        end
(5) headrl := 0
```

Figs 1 and 2 show an example illustrating the algorithm. Fig. 1 shows an input binary image. Fig. 2 is the labeled image using our algorithm. Equivalent pairs generated at line 4 are (2, 1), (3, 4), (1, 3), while at line 7 is (5, 6) and at line 8 is (4, 6).

Figures 3(a)-3(h) show successive applications of Algorithm 1 after processing the equivalent pairs generated at the end of each line.

III Analysis of the algorithm

Let N be the height of the image and M the length of the image. We claim that the algorithm takes no more than O(N × M). Let us define E_k to be the content of table E after execution of Algorithm 1 at the end of labeling line k. If no equivalent pair is generated during line k, then E_k=E_{k-1}. We may denote the original table as E_0. Define sets of labels T(1), T(2),...T(M) as follows.

Let T(1) = { x | E_0[x] = x }.

Define T(i) = { x | x in T(i-1) and E_{i-1}[x] = x }
for all 1<= i<=M

Lemma 1

For all 1 ≤ k ≤ M, the following holds:

For all x in T(k):
if E_k[x] = y and y ≠ x, then E_k[y] = y (*)

Proof

We prove this by induction on k.

Basis:

Let k = 1.

x is in T(1) iff E_0[x] = x. Now E_0[x] = x, for all x. No labeled pair can be generated during the labeling of the first line. Hence E_1[x] = x; Hence (*) holds vacuously.

Inductive step:
Assume that the lemma is true for k - 1.

After labeling line k, if no equivalent pair is generated, then E_k = E_{k-1} and T(k) = T(k - 1). Hence for all x in T(k), (*) holds by inductive hypothesis.

Now suppose q pairs are generated during the labeling of line k. The only changes made to table E is by Algorithm 1 processing each pair. In the following, use E' to denote the current content of table E. That is, at the beginning of labeling line k, E' = E_{k-1}.

Let (x,y) be the first merge labels from which the first equivalent pair is generated. That is E'[x] ≠ E'[y]. Let E'[x] = x', and E'[y] = y'. The equivalent pair generated is (x', y'). By the inductive hypothesis, E'[x'] = x' and E[y'] = y'. Let SIZE[x'] = 1 and SIZE[y'] = 1 (see Algorithm 1). Algorithm 1 changes E' by setting E'[x'] to y', hence (*) still holds if we replace E_k by E' in (*). We can show that for each pair that Algorithm 1 processes, it changes E' in a

way that (*) always holds. Let (a,b) be the pair of merge labels pairs from which the i-th equivalent pair is generated. Let E'[a] = a' and E'[b] = b', i.e. (a', b') is the equivalent pair generated. Assuming that (*) is true for E' after i-1 pairs, E'[a'] = a' and E'[b'] = b'. Without loss of generality, assume SIZE[a'] < SIZE[b']. Algorithm 1 changes E such that E'[a'] = b' and for all z s.t. E'[z] = a', E'[z] is set to b'. Since E'[b'] = b', (*) holds for E' after (a',b') is processed. After all q pairs are processed, E' is E_k. Hence (*) holds for k.

□

Lemma 2

After execution of Algorithm 1 at the end of each line, say k, all E_k [x] such that E_k [x] ≠ x will never be changed by subsequent execution of Algorithm 1.

Proof

Let E_k[x] = y and y ≠ x. In the labeling process, whenever x appears as any merge label in line t, where t>k the current x entry in E i.e. E[x] =y will be used for labeling. If an equivalent pair is to be generated at this point, y will be used. Now E_k[y] = y then y is a representative element. Then SIZE[y] will be reset to 1 in step III. Hence the NEXT component of y will be given new values in steps (I), (II) in the next execution of the Algorithm 1. Hence all E[x] s.t. E_k[x] = y and y ≠ x will never be updated subsequently.

□

Theorem

The labeling and merging Algorithm takes time O(N × M) where N × M is the size of the input binary image.

Proof

It is clear that the labeling portion of the algorithm takes no more than constant time for each pixel. Hence the time is 0(N × M). We need to analyze the time taken for each application of Algorithm 1. Let us assume that q pairs are generated during one line to be processed by Algorithm 1. By Lemma 1, it takes constant time to determine for each pair (x,y) generated whether currently E[x] = E[y]. Each time a relation E[x] ≠ E[y] is discovered, changes in the table E takes time proportional to the size of the smaller set in question. Let the sum over all such times be S. By lemma 2, S is dependent only on the pairs generated in this line and not on the pairs generated in subsequent lines. In the worst case, S is O(q log q) for q equivalent pairs generated [7].

The time taken to reset SIZE and START of elements on the r-list will be proportional to q i.e. O(q).

We shall consider the complexity of the pairs generated in relation to the size of the image it takes to generate them. We need only consider pairs generated on one line as those generated on previous line would have been considered by previous execution of Algorithm 1.

Let m be the number of pairs generated in one line. For m = 2, the SIZE of any set involved in step (2) of Algorithm 1 is 1. For m = 3, Fig. 4 shows the four ways, when a minimum size image can generate pairs for which the maximum number of E entries need to be changed.

The smallest image by which m = 7 pairs are generated and which has the maximum number of E entry updates by Algorithm 1 is shown in Fig. 5.

Algorithm 1 process each pair (x,y) by appending the shorter list onto the longer list. The maximum complexity for the Algorithm occurs when the two lists are always of equal length. In the case with 4 labels, it would involve three equivalent pairs, as illustrated in Fig. 4. It is clear that it needs at least 4 lines to generate this patterns. In the case of 8 labels involving 7 pairs, it needs at least 2 more lines. To generate m equivalent pairs on one line with maximum complexity, an image of the size having a minimum of O(m) pels in the x direction and a minimum of O(log m) pels in the y direction is required. The total area needed to generate this image is O(m log m). This area must be connected, but holes may exist as illustrated by Fig. 6. The entire image area can be considered to be made up of connected areas each having sides parallel to either the x-axis or the y-axis. The sum over all executions of Algorithm 1 is proportional to the total sum of these areas, hence the total size of the image.

As a consequence of lemma 2, for any q equivalent pairs generated, at most O(q log q) time is needed to execute Algorithm 1. This worst case occurs when each pair (x,y) is being processed, the list containing x and that containing y are of equal length. This q pairs need at least O(q log q) area to generate

as discussed above. Hence in this case the time taken to execute Algorithm 1 is proportional to the size of the area needed to generate q pairs. A second case need to be considered. This is when O(N) pairs is generated on the same line, needing only two lines. This need an area of only O(N) to generate. However, processing each pair need only constant time, as the shorter list referred in Algorithm 1 would always be containing only one element. Since Algorithm 1 is executed at the end of each line, these 0(N) pairs generated on one line can be processed in O(N) time. Again, the execution of Algorithm 1 is proportional to the area generating these pairs. Given any image, it can be divided into areas where equivalent pairs whose labels belong to the same list are generated. Summing over all area of the image, the total time for executing Algorithm 1 is O(N × M).

□

IV Optimality

In general the above algorithm generates less equivalent pairs than previously known algorithms. In the following, we shall illustrate label images with only the merge labels. So far we have only discussed the use of merge labels. We shall discuss the use of the feature labels later in the section. Consider an image shown in Fig. 6(a): Fig. 6 (b) shows the labels as obtained using previous algorithms such as in [3].

For an input image of size N × M, the number of distinct labels is O(M) and the number of pairs generated is O(N × M).

Fig. 6 (c) is the image labeled by the algorithm given in Section II.

The number of labels is O(M), while the number of pairs generated is O(M) also, an order of magnitude improvement.

However, one can construct an image such that using our algorithm, the number of pairs generated for an image of size N × M is still O(N × M). Figs. 7 (a) and 7 (b) show an input image and a labeled image using our algorithm where O(N × M) distinct labels are used and O(N × M) pairs are generated. Notice that this asymptotic behaviour is still true even if we use larger window or finite lookahead in our labelling algorithm.

Given any window of constant size, say k × k, it is always possible to construct a worst case image such that the local information in the window is not sufficient to determine connectivity that can only be discovered later through label merging. Naturally, with a larger size window, fewer equivalent pairs will be generated by some constant factor. However, the asymptotic complexities of the number of labels and number of equivalent pairs are still the same.

Since labeling in raster scan fashion takes at least 0(N × M), our algorithm combining both labelling and merging achieves this bound and hence is asymptotically optimal.

V Discussion

Our proposed algorithm combines labeling and merging and takes total time no worse than O(N × M) for an image of size N × M. The steps taken by Algorithm 1 can also be executed as soon as each pair is generated. This will further reduce the number of equivalent pair generated, as demonstrated by the example in Figs. 8(a) and 8(b).

If (1,2) in Fig. 8(a) is processed by Algorithm 1 as soon as it is generated, the labelled image will be as in Fig. 8(b) and the new pair (2,1) will not be generated.

In some real time applications, component labeling is executed by special hardware. In this case, Algorithm 1 can be overlapped with labeling. As each pair is generated Algorithm 1 can proceed to process it. It is clear that any update to table E does not affect the correctness of the labeling algorithm, since whenever E(x) is replaced, it is only replaced by an equivalent merge label. In most application, equivalent pair will not be generated every possible cycle. A buffer for the labeled pairs of at most N/2, (depending on the size of the window) is needed for use by Algorithm 1. In cases when Algorithm 1 is finished processing when labeling of a line is done, the labeling process can continue without stopping. The labeling need only be stopped in cases where Algorithm 1 still have pairs to process. When labeling is suspended, Algorithm 1 may also be overlapped with a purging algorithm such as described in [13] to recover reusable feature memory locations if desired. If the pairs generated are not all on one line and not in complex relation as illustrated in the previous sections, it is possible that the labeling process need never stop.

Note that the above algorithm labels all consecutive 1's on the same row by the same feature label. In fact, given any pair of pixels x,y with feature labels F,

there is a path of adjacent pixels from x to y such that all feature labels of pixels on the path is F. This may not be true for the merge labels. That is, some regions with the same merge labels may not be connected in the sense that there may not be a path of pixels all labeled with the same merge label connecting the two regions in question. However, given any two pixels a,b labeled with merge labels m, there exists a path of adjacent pixels from a to b such that all merge labels of pixels on the path is m or some label equivalent to m. In some region analysis applications, various features such as area, perimeter, centroid may be kept on line and updated as each pixel is labeled. In this case, only two label line buffers are kept, not the labels of the entire image. For a feature such as area of an object, one may update the feature according to the merge label and the result of combining all features of the labels that are found to be equivalent would still be correct. This is because the area of an object containing a merge label m is simply the sum of all the pixels with merge label m and its equivalent. In this case, the algorithm need only label each pixel with the merge label. The use of different labels in partial feature information collection will not be discussed in detail here.

VI Conclusion

Component labeling algorithm of binary images are important in many machine vision applications. Component labeling algorithms based on a raster scan of the image using a fixed size window are attractive for hardware implementation. Traditionally the image is being labeled and equivalent pairs are generated and a second merging phase is performed to merge the equivalent labels.

In this paper, we discuss an algorithm which combines the labeling with the merging process. A significant result is that the algorithm takes $0(N \times M)$ where $N \times M$ is the size of the image and is asymptotically optimal.

References

[1] R.T. Chin, "Algorithms and Techniques for Automated Visual Inspection", in Handbook of Pattern Recognition and Image Processing, ed. by T.Y. Young . K.S. Fu, Academic Press, 1986, pp. 587-612.

[2] R.T. Chin . C.A. Harlow, "Automated Visual Inspection: a survey", IEEE Transactions on Pattern Analysis and Machine Intelligence", PAMI-4, No. 6 (Nov. 1982), pp. 557-573.

[3] Foith, et al, "Real-time Processing of Binary Images for Industrial Applications", in "Digital Image Processing Systems", L. Bolc and Z. Kalpa, editors, Springer Verlag, Berlin, 1981.

[4] D. Petkovic et al, "An experimental system for disc heads inspection", Proc. of the 8th Int. Conf. on Pattern Recognition, Oct., 1986, Paris, France.

[5] R.C. Gonzalez and P. Wintz, Digital Image Processing, Addison Wesley, 1987.

[6] Rosenfeld, A. and Kak, A., "Digital Picture Processing", 2nd edition, Vol. 1, 2, Academic Press, 1982.

[7] Ballard, D.H. and Brown, C.M., "Computer Vision", Prentice-Hall, Englewood Cliffs, N.J., 1982.

[8] Berman, S., Parikh, P. and Lee, G.C.S., "Computer Recognition of Overlapping Parts using a single camera", Proc. Pattern Recognition & Image Processing, June, 1982, pp.650-655.

[9] Lumia, R., Shipiro, L. and Zuniga, O., "A new connected components algorithm for virtual memory computers", Proc. Pattern Recognition & Image Processing, June, 1982, pp.560-565.

[10] Wahl, F.M., Wong, K.Y. and Casey, R.G., "Block Segmentation and Text Extraction in Mixed Text/Image Documents", IBM Research Report, IBM San Jose Research Report, 1982.

[11] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

[12] Tarjan, R.E. "On the efficiency of a good but not linear set merging algorithm", JACM, 1975,pp. 215-225

[13] Fong, Amelia, C., "Algorithms for realtime component labeling of Images", Vision and Image Computing, Vol. 6, Feb., 1988, pp.21-27.

[14] Cypher, R. and J.L.C. Sanz, "SIMD architectures and algorithms for image processing and computer vision," IEEE Trans. ASSP, Vol 37, Dec., 1989, pp.2158-2174.

[15] Cypher, R., Sanz, J.L.C. and Snyder, L., "Hypercube and Shuffle-exchange algorithms for image component labeling," J of Algorithms, Vol. 10, 1989, pp.140-150.

[16] Tarjan, R.E., and Van Leeuwen, J., "Worst-case analysis of set union algorithms", JACM 31, pp.245-281.

[17] Z. Galil and G.F. Italiano, "Data structures and Algorithms for Disjoint Set Union Problems", ACM Computing Survey, Sept., 1991, pp. 319-344.

```
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 1 1           1
          1 1 1
```

Fig. 1

```
1 1 1 1 1 1 1 1 1 1
1                   1
1   2         3     1
1 1     4 4     3
                    4
        5 5 5       4
    6 6 6           4
          6 6 6
```

Fig. 2

	E	SIZE	START	NEXT	REPF	REPB
1	1	1	1			
2	2	1	2			
3	3	1	3			
4	4	1	4			
5	5	1	5			
6	6	1	6			
	.	.	.	.		
	.	.	.	.		

Fig. 3(a) Initial tables

	E	SIZE	START	NEXT	REPF	REPB
1	1	2	2	0	0	4
2	1	1	2	1		
3	4	1	3	4		
4	4	2	3	0	1	0
5						
6						
	.	.	.	.		
	.	.	.	.		

headrl = 4

Fig. 3(b): After processing (2,1) (3,4)

	E	SIZE	START	NEXT	REPF	REPB
1	4	2	2	3		
2	4	1	2	1		
3	4	1	3	4		
4	4	4	2	0	0	0
5						
6						
	.	.	.	.		
	.	.	.	.		

headrl = 4

Fig. 3(c): After processing (1,3)

	E	SIZE	START	NEXT	REPF	REPB
1						
2						
3						
4	4	1	4			
5						
6						
	.	.	.	.		
	.	.	.	.		

headrl = 0

Fig. 3(d): After resetting at the end of line 4

	E	SIZE	START	NEXT	REPF	REPB
1						
2						
3						
4	4	1	4			
5	6	1	5	6		
6	6	2	5	0	0	0
	.	.	.	.		
	.	.	.	.		

headrl = 6

Fig. 3(e): After processing (5,6) at line 7

	E	SIZE	START	NEXT	REPF	REPB
1						
2						
3						
4	4	1	4			
5						
6	6	1	6			
	.	.	.	.		
	.	.	.	.		

headrl = 0

Fig. 3(f): After resetting at the end of line 7

	E	SIZE	START	NEXT	REPF	REPB
1	4					
2	4					
3	4					
4	4	1	4	6		
5	4					
6	4	2	4	0	0	0
	.	.	.	.		
	.	.	.	.		

headrl = 6

Fig. 3(g): After processing (4,6) at line 8

	E	SIZE	START	NEXT	REPF	REPB
1	4					
2	4					
3	4					
4	4					
5	4					
6	4	1	6			
	.	.	.	.		
	.	.	.	.		

headrl = 0

Fig. 3(h): After resetting at the end of line 8

```
1 1 1 1 1 1 1 1 1
1                 1
1   2         3   1
1 1     4 4   3
```

```
1 1 1 1 1 1 1 1 1
1                 1
1   2         3   1
1 1     4 4 4 4
```

```
    1 1 1 1 1 1 1
    1           1
    1       2   1
3 3     4 4   2
```

```
    1 1 1 1 1 1 1
    1           1
    1       2   1
3 3     4 4 4 4
```

Fig. 4

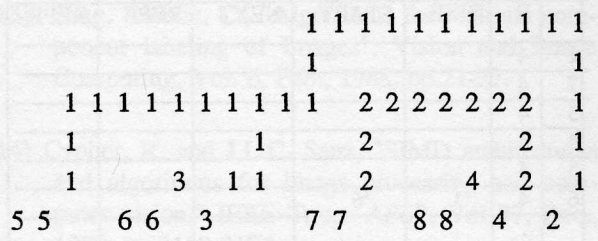

Fig. 5

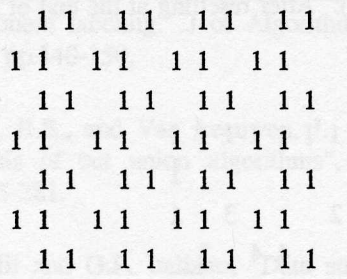

Fig. 6 (a)

Fig. 6 (b)

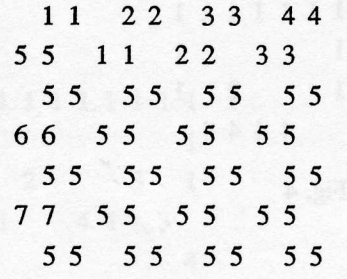

Fig. 6 (c)

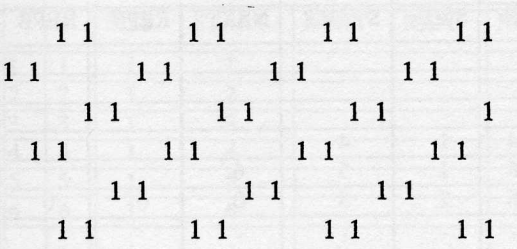

Fig. 7 (a)

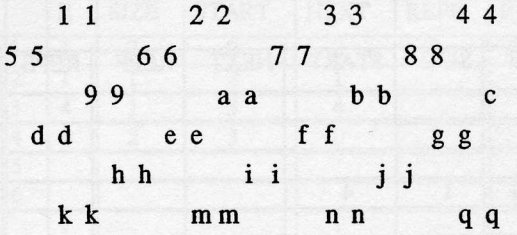

Fig. 7 (b)

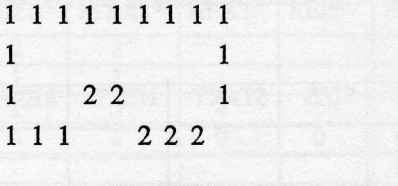

Fig. 8 (a)

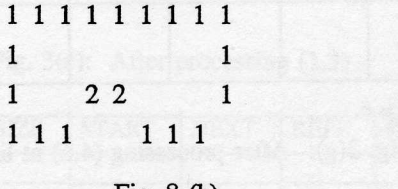

Fig. 8 (b)