Union-Retire: A New Paradigm for Single-Pass Connected Component Analysis

Donald G. Bailey\textsuperscript{1} and Michael J. Klaiber\textsuperscript{2}

\textsuperscript{1} Centre for Research in Image and Signal Processing, Massey University, Palmerston North 4442, New Zealand
D.G.Bailey@massey.ac.nz

\textsuperscript{2} Computer Science, Baden-Wuerttemberg Cooperative State University (DHBW), 70178 Stuttgart, Germany
michael.klaiber@lehre.dhbw-stuttgart.de

Abstract. Most connected component labelling and analysis algorithms are based on some variant of Union-Find. In this paper, it is shown the Find operation is unnecessary for single-pass algorithms, leading to the Union-Retire approach where the focus is on connectivity rather than labelling. The computational complexity of the resulting algorithm is linear with the number of pixels within an image. It is shown that the resulting algorithm requires significantly less memory than existing algorithms, making it ideal for embedded image processing.

Keywords: Union-Find · Connected components · Feature extraction

1 Introduction

Connected component analysis (CCA) is a fundamental operation in many image analysis and machine vision tasks. The purpose of CCA is to derive the feature vector of each set of connected object pixels within a binary image.

Prior to CCA, an input image must be preprocessed to detect the objects of interest and to segment the objects from the background. This results in a binary image, with each connected set of detected pixels corresponding to an object. Then a connected component labelling (CCL) algorithm is commonly used to assign a unique label to each connected component. Finally, a feature vector is extracted for each object, using the component labels to distinguish pixels associated with different objects.

Direct CCA algorithms combine the labelling and feature extraction steps, directly producing the set of feature vectors from a binary image. Consequently, the labelled image is not actually required, but is only used as an intermediate data structure to aid in calculating the feature vector for a completed component.

Most CCL and CCA algorithms are based on some form of Union-Find data structure and algorithm [11,19]. Typical CCL algorithms require two raster-scan
passes through the image. During the first pass, preliminary labels are assigned to each pixel by propagating labels from adjacent already processed object pixels. If an object pixel has no labels within its processed neighbourhood then a new label is assigned. If an object pixel has two different preliminary labels within its neighbourhood, for example where sub-components join at the bottom of a “U” shaped object, then these labels represent the same object and the labels, or sets of labels, from the two branches are then combined. This is a Union operation, with the resultant set containing all of the preliminary labels that represent a single connected component. To obtain a consistently labelled image, a second pass is necessary to relabel each pixel with the representative label from the set associated with a connected component. The Find operation returns the representative label from the set containing a given preliminary label.

In contrast, single-pass CCA only processes each pixel once. This requires extracting the feature vector during the labelling process [1,2]. When regions merge (during a Union) it is necessary to also combine the associated feature vectors. This limits the components of feature vectors to those that can be accumulated and combined associatively, for example: area, bounding box, centre of gravity, best-fit ellipse, perimeter, average pixel value, average colour [5]. Single-pass CCA also requires a Find operation for each object pixel to ensure that the data for each pixel is accumulated into the correct feature vector. Union and Find operations are therefore intermixed depending on the particular arrangement of object pixels within the image being processed.

1.1 Union-Find Algorithms

The data structures which represent equivalences (as a result of a Union) between preliminary labels are often mapped to arrays, where the label is the index into one or more arrays. These arrays are used to represent a directed acyclic graph (DAG), with the nodes within a connected graph representing the preliminary labels assigned to a single connected component. With a single array, commonly each node has only a single arc leaving it. Example graphs and their corresponding array representations are shown in Fig. 1.

![Fig. 1. Graphs of preliminary labels associated with a connected component and their array representation. The shaded node corresponds to the representative label. (a) General case. (b) Quick-Find.](image)
A *Find* operation follows the arcs to determine the current representative label for a component. This may require several look-ups within the associated table, in the general case (Fig. 1(a)). The *Quick-Find* variant of *Union-Find* [6] arranges the graph so that all nodes point directly to the representative label (Fig. 1(b)), enabling the representative label to be found with a single lookup.

For a *Union* operation, it is necessary to first *Find* the representative label of each set, and then link the trees if these are different. Consider a *Union* between nodes 2⃝ and 4⃝ in Fig. 2; the representative labels are nodes 1⃝ and 3⃝ respectively. The *Quick-Union* algorithm [6] directly links node 3⃝ to 1⃝ (Fig. 2(b)) requiring only one additional operation. However, within *Quick-Union* a *Find* may require multiple lookups. To enable *Quick-Find*, when performing a *Union* operation it is necessary to find all the nodes linked to the old representative label (3⃝) and link these directly to the new representative label (1⃝ in Fig. 2(c)).

Both *Quick-Union* and *Quick-Find* have quadratic computation complexity in the worst case [15].

This may be partially overcome by using *Quick-Union* with path compression. Whenever a *Find* operation is performed, all of the nodes visited in the search from the initial node to the representative node are linked directly to the representative node so that subsequent searches on that path are faster. The complexity of *Quick-Union* with path compression grows with the inverse Ackermann function [19], which is only slightly worse than linear.

With CCL, the focus is the algorithms is on the labelling process, as the goal is to provide a labelled image. This also carries over to CCA algorithms, which have as CCL as their root, even though a labelled image is not actually required. This stems from the underlying *Union-Find* algorithm, where the goal of the *Find* operation is to determine the representative label of the set. To overcome this, we propose a new approach which focuses on connectivity between sub-components, rather than the labels. *Union-Retire* does not need the *Find* operation, and this is replaced with *Retire* when a node is no longer required.

### 1.2 Contributions

The key contributions of this paper are the algorithmic description and analysis of the novel *Union-Retire* algorithm.
The remainder of the paper is structured as follows: Sect. 2 explores earlier work on single-pass CCA algorithms in terms of the underlying Union-Find algorithms used. Then a complete algorithmic description of Union-Retire is provided in Sect. 3. The correctness of the new algorithm is analysed in Sect. 4, and the complexity is compared with other single-pass CCA algorithms, before giving the conclusions in Sect. 5.

2 Prior Work

Klaiber et al. [11] analysed several CCL and CCA algorithms in relation to their underlying merging methods in terms of Union-Find. Here, we build on and extend this work in the context of single-pass CCA algorithms. A recent review investigated the history and development of single-pass CCA algorithms [5]. Here, the focus will be on the underlying variation of Union-Find that is used by each single-pass CCA algorithm.

Early work by Selkow [16] and Veillon [23] used a set of finite state automata, one per column, to process the image one line at a time. The representative label of each component was the column of the leftmost pixel within the component on the current row, which is unique for each component. This definition results in the representative label changing with each row in the image. The image was processed one row at a time, with Selkow considering single pixels and Veillon runs of consecutive pixels. Overlaps between successive rows caused the propagation of labels from the previous row along the current row. There is no explicit Union, although overlapping runs from one row to the next are used to determine which component a pixel or run belongs to. The label propagation along the row is effectively performing a Find for each pixel on a row.

The early work by Bailey [1], although more region growing than CCA, did extract feature vectors in a single pass. This clearly used Quick-Find, with all of the old links immediately updated to the new representative label. Consequently, a Union was relatively expensive (quadratic complexity), especially when there was a chain of mergers, which require the same labels to be updated multiple times.

Trein et al. [21] used run-length encoding, and then processed a run of consecutive object pixels at a time, propagating the representative label from adjacent runs in the previous row. A single pointer from the old label to the representative label is added for a Union operation (Quick-Union). The Find is not explicitly described, but would require multiple lookups to determine the representative label.

Most of the remaining algorithms process a pixel at a time (rather than runs) and are based on the early work by Bailey and Johnston [2,8]. These use a Quick-Union algorithm with path compression deferred to the end of the row, enabling a single lookup to be used to determine the representative label. Chains of mergers (a sequence of Union operations) do not affect the processing of the remainder of the current row, since the old labels are not accessible again until the following row. Affected pairs of labels are pushed onto a stack, enabling them
to be quickly revisited in the reverse order at the end of the row to relink them to their respective representative label (context-based path compression). The overhead of performing this path compression is typically 1–2%, although could be 20% of the image width in the worst case.

Klaiber et al.’s [9,10] refinements to Bailey and Johnston’s algorithm mainly improved memory efficiency by recycling labels that were no longer used within the processing. The underlying Union-Find algorithm is the same. Klaiber et al. [11] also showed that it was only necessary to perform a double lookup of the first pixel in a run, and use that label for all pixels in the run. This is a minor variation on Quick-Union with context-sensitive path compression.

By processing rows in zig-zag order [4], path compression is handled automatically when processing the following line. This enables Quick-Union with a single lookup to be used for the Find operation without any end of row overhead.

Spagnolo et al. [17] used Quick-Union, with a triple lookup for Find. While this is successful for many practical images, three lookups may not necessarily reach the representative label. Such cases must be resolved at the end of the frame. Similarly, Thornberg and Lawal [20] and Malik et al. [13] perform a simple Union and defer merger resolution until the end of the frame.

Aggressive relabelling by Ma et al. [3,12] reassigns labels for each row processed. This effectively requires two lookups: the first to determine the representative label in the previous row, and the second to translate that to the corresponding representative label used in the current row. The algorithm uses Quick-Union with context-based path compression at the end of each row. However, since some mergers are implicitly managed through the translation from one row to the next, the worst case overhead reduced to 6.3% [11,12].

A modification by Zhao et al. [26] defers assigning a label until the end of a run of successive object pixels. This can reduce the number of trivial mergers, and reduce the depth of the resulting tree. Zhao incorrectly claimed that this eliminates the need for path compression at the end of each row, however a single lookup is not sufficient to give the correct representative label. Therefore, this Quick-Union without path compression is insufficient, and consequently, the feature vectors are likely to be incorrect for some patterns.

Jeong et al. [7] took a different approach. For the Union operation, they replaced all of the old labels directly with the representative label in parallel using hardware. Thus only one label us used for a connected component at any time, effectively a form of Quick-Find.

The next two papers reviewed perform run-based labelling, but check for mergers using a 2 × 2 pixel window. Zhao et al. [25] assign a sequential index for each run within a row, and a serial number of the run within the image. The image-based serial number is used as the representative label, whereas the row-based index is used to address the feature vector and the mapping to the serial number. When runs in successive rows overlap, a form of Quick-Union is used to identify equivalences from one row to the next, with the process of merging feature vectors and resolving equivalences deferred until the end of each row. The merger resolution process has quadratic computational complexity.
Tang et al. [18] represent the Union-Find data structure using a linked-list. This is implemented using 3 tables: head, next, and tail. The head is the current representative label, and changes dynamically as labels expire. Data is accumulated in the representative label. When runs in successive rows overlap, the Union operation combines the lists, and provides translation of labels between rows. The head and tail tables also enable the Find operation in constant time.

The focus of all of these algorithms is primarily on the labelling of pixels or runs, with the feature vector accumulation almost a by-product of the processing. This is perhaps a little less so in Tang et al. [18], which is the most similar to our proposed approach. Union-Retire removes the focus on any particular node as being the representative node, and in so doing relaxes the required representation.

### 3 Union-Retire Algorithm

Like Zhao et al. [25] and Tang et al. [18], each run of consecutive object pixels along a row is sequentially assigned a unique index. The basic principle is to represent each run as a node within a DAG where each connected graph corresponds to a single connected component. Within this description, the terms node, run, and index are used interchangeably.

When runs overlap between successive rows, a Union operation links the associated nodes. This link is represented by an arc within the DAG from the earlier index to the later index.

Since the focus is shifted from labelling, a Find operation is unnecessary, since there is no representative label or node. Linking is therefore simplified, with the graph no longer constrained to be a tree structure. As long as the set of nodes within a component are connected, the underlying DAG may have several roots and leaves.

When a run of pixels is no longer accessible (due to the raster scan), a Retire operation removes the corresponding node from the DAG, simplifying the graph’s structure. Any feature data associated with a retired node is then accumulated into one of its children, and the graph adjusted to maintain connectivity. Finally, when the last node within a component is Retired, the connected component is completed, and the associated feature vector is output.

Thus, the primary attention of Union-Retire is on adding (Union) and removing (Retire) nodes from a graph, rather than on the incidental task of finding the representative node or label. Throughout this process, the focus is on maintaining the overall connectivity between the constituent runs within the component.

### 3.1 Data Structures

The DAG is represented as an array, indexed by the node index. Each node can have up to two outgoing arcs, with the destination node for each arc are stored in one of two link arrays, \( L1[ ] \) and \( L2[ ] \).

Each node also has associated feature vector data, stored in the data array, \( D[ ] \). This includes the feature vector derived from the run of pixels, plus any data passed from retired nodes.
Algorithm 1 Union-Retire CCA algorithm

Input: Binary image $I$ of width $W$ and height $H$
Output: A feature vector for each connected component in $I$

1: $R_p := R_c := 0$ \Comment{Initialise run indices}
2: for $y := 0$ to $H$ do
3: \hspace{1em} for $x := 0$ to $W$ do \Comment{Scan}
4: \hspace{2em} Window := $I[x-1 : x; y-1 : y]$ \Comment{Form 2\times2 window}
5: \hspace{2em} if Window = \textit{No links} then \Comment{Start of run in previous row}
6: \hspace{3em} $R_p += 1$
7: \hspace{2em} end if
8: \hspace{2em} if Window = \textit{Two links} then \Comment{Start of run in current row}
9: \hspace{3em} $R_c += 1$
10: \hspace{2em} end if
11: \hspace{2em} $L1[R_c] := L2[R_c] := 0$ \Comment{No links yet}
12: \hspace{2em} $D[R_c] := FV(x, y)$ \Comment{Feature vector for pixel}
13: \hspace{2em} else if Window = \textit{Only one link} then \Comment{Continuing current run}
14: \hspace{3em} $D[R_c] += FV(x, y)$ \Comment{Accumulate feature vector}
15: \hspace{2em} end if
16: \hspace{2em} if $L1[R_p] = 0$ then \Comment{Union, link nodes}
17: \hspace{3em} $L1[R_p] := R_c$
18: \hspace{2em} else if $L2[R_p] = 0$ and $L1[R_p] \neq R_c$ then \Comment{Only one link}
19: \hspace{3em} $\text{LINK}(L1[R_p] \rightarrow R_c)$
20: \hspace{3em} $L1[R_p] := R_c$
21: \hspace{2em} else if $L1[R_p] \neq R_c$ and $L2[R_p] \neq R_c$ then \Comment{Two links}
22: \hspace{3em} $\text{LINK}(L2[R_p] \rightarrow R_c)$
23: \hspace{3em} $L2[R_p] := R_c$
24: \hspace{2em} end if
25: \hspace{2em} end if
26: \hspace{2em} if Window = \textit{Two links} then \Comment{Retire, unlink node}
27: \hspace{3em} $\text{output: } D[R_p]$ \Comment{Object completed}
28: \hspace{2em} else if $L2[R_p] = 0$ then \Comment{Only one link}
29: \hspace{3em} $D[L1[R_p]] += D[R_p]$ \Comment{Accumulate data}
30: \hspace{2em} else \Comment{Two links}
31: \hspace{3em} $D[L2[R_p]] += D[R_p]$ \Comment{Accumulate data}
32: \hspace{3em} $\text{LINK}(L1[R_p] \rightarrow L2[R_p])$ \Comment{Maintain linkages}
33: \hspace{2em} end if
34: \hspace{2em} end if
35: \hspace{2em} end if
36: \hspace{2em} end for
37: end for
38: \textbf{procedure} $\text{LINK}(R_a \rightarrow R_b)$ \Comment{Link two nodes}
39: \hspace{2em} if $L1[R_a] = 0$ then \Comment{No links}
40: \hspace{3em} $L1[R_a] := R_b$ \Comment{Add link}
41: \hspace{2em} else if $L2[R_a] = 0$ and $L1[R_a] \neq R_b$ then \Comment{Only one link}
42: \hspace{3em} $L2[R_a] := R_b$ \Comment{Add second link}
43: \hspace{2em} else if $L1[R_a] \neq R_b$ and $L2[R_a] \neq R_b$ then \Comment{Two links}
44: \hspace{3em} $\text{LINK}(L2[R_a] \rightarrow R_b)$ \Comment{Pass link on}
45: \hspace{2em} end if
46: \hspace{2em} end procedure
Within the algorithm description below, the indices are simply sequential. However, for a given image width, $W$, there can be at most $\left\lfloor \frac{W}{2} \right\rfloor + 2$ indices in use at any one time. In practise, the run index is provided by a counter that can wrap around, and this limits size of the memory arrays required.

### 3.2 Algorithm Description

The processing steps for `Union-Retire` are listed in Algorithm 1.

The image is scanned in raster fashion, using a $2 \times 2$ local window to determine connectivity (8-connectivity is assumed here, but can be trivially changed for 4-connectivity). For simplicity, the handling of the edge of the image is ignored here by processing one extra pixel in each row, and one extra row of the image, with the assumption that pixel values outside the image are background pixels. In practise, by handling the border cases appropriately, it is only necessary to process each pixel without adding extra rows or columns. The bottom right corner of the $2 \times 2$ window corresponds to the current pixel. Within the algorithm, the window is represented by a $2 \times 2$ graphic, where white represents background, black represents an object pixel, and grey is a “don’t care” (either object or background).

While the run indices are shown as unique in Algorithm 1 (the index is incremented for each successive run), in practise, the indices wrap, with the corresponding entries within the memory arrays reused. Since these are both assigned and retired sequentially, it is unnecessary to maintain a queue of recycled indices. Let $R_p$ and $R_c$ be the most recently used indices on the previous and current rows respectively. These are updated (incremented) when a new object pixel is found in the corresponding row (previous row: $[\text{\[\]}$, line 5; current row: $[\text{\[\}}$, line 8).

The feature vector (stored in $D[\]]$) for the node on the current row is initialised at the start of a run ($[\text{\[\]}$, line 8), and updated for each object pixel within a run on the current row ($[\text{\[\}}$, line 12).

When a run in the previous row is connected to a run in the current row, a `Union` operation is invoked, which adds a link between the corresponding nodes to connect the two sub-components. This is detected either at the end of the run on the current row when there is an object pixel in the previous row, or at the end of a run on the previous row when there is an object pixel in the current row ($[\text{\[\]}$, line 15). This link is always added from an earlier node to a later node to facilitate the `Retire` operation. If the node on the previous row already has link, then the link is added to its successor, so that several successive nodes are linked in a chain (see `Union(1,4)` and `Union(1,5)` in the following example). If the successor node already has two links, then the link is propagated to the successor’s most recent successor (see `Union(6,11)` and `Union(7,12)` in the following example).

At the end of a run on the previous row ($[\text{\[\]}$, line 26), the associated index is never accessed again. Therefore, a `Retire` operation is performed, removing the corresponding node from the graph. The feature vector associated with the
retired node is accumulated into a linked node (if any). If there are no linked nodes, then the object is detected as completed, and the feature vector of the connected component is output. If there are two linked nodes, then it is necessary to insert a link from the earlier node to the later node to maintain overall connectivity within the DAG.

3.3 Example

The sequence of Union and Retire operations for an example image is shown in Fig. 3, with the associated directed graphs after each operation. The key features of the algorithm will be highlighted using this example.

At pixel (6,2) the first Union operation links nodes 1⃝ and 3⃝. The link is always from an earlier node to a later node.

At (9,2) when nodes 1⃝ and 4⃝ are linked, a link is inserted between nodes 3⃝ (the current child of 1⃝) and 4⃝, and the node on the previous row is updated so that it points to the node just inserted. This is so that a sequence of Unions from the same source run can be efficiently processed without requiring a large number of links from the source node. Note that although there is no longer a direct link between nodes 1⃝ and 3⃝, they are still implicitly connected via 4⃝.

Then, when node 5⃝ is linked at (10,2), it can simply be added at the end of the chain. Note that the Union operation is performed before node 1⃝ is Retired. When 1⃝ is Retired, its data is combined into 5⃝, and it is removed from the graph. 1⃝ is no longer accessible and plays no further part in the operations. However, the effects of its connectivity with each of nodes 3⃝, 4⃝ and 5⃝ are represented in the links between those nodes.

At pixel (4,3), the connection between 3⃝ and 7⃝ links the two constituent sub-components. The child of 3⃝ is 4⃝, so the link to 7⃝ is added to 4⃝. It does not matter that 4⃝ is not the end of the chain; node 4⃝ now has 2 links.

Similarly, when 4⃝ is linked to 9⃝, 9⃝ is added as a second link to 7⃝. Node 5⃝ remains connected to 4⃝. When 4⃝ is Retired at (9,3), it has two child nodes: 5⃝ and 9⃝. To maintain connectivity, it is necessary to link from 5⃝ to 9⃝.

At (2,4) when 6⃝ is linked to 11⃝, 6⃝’s child 7⃝ already has 2 children, so the link table for 7⃝ is full. Therefore the link is propagated to 7⃝’s most recent child, and is added to 9⃝. Similarly at (4,4) when 7⃝ is linked to 12⃝.

At the bottom of the loop, at pixel (8,5) with Union(13,14), nodes 13⃝ and 14⃝ are already linked, so no update is required to the DAG.

Finally, when 14⃝ is Retired at (8,6) it has no children. Therefore the connected component is complete, and the associated feature vector can be output. Note that this is the earliest possible time that completion could be detected.

4 Analysis

4.1 Correctness

There are two aspects to the correctness of the algorithm that need to be demonstrated. The first is that each pixel in a connected component is assigned to the
Fig. 3. Example image showing linkages between runs after each Union and Retire step. Node numbers correspond to run indices.
correct component, and the second (related to the first) is that the correct feature
is calculated for each connected component.

Each node in the DAG is associated with an index. Each pixel is assigned
an index with all consecutive object pixels within a run assigned the same node.
Incrementing the index on the first pixel of a run is sufficient to guarantee that
each node has a unique index. When recycling indices, it is essential that the
index is retired before it is assigned to another run. This is satisfied by having
at least \( \left\lfloor \frac{W}{2} \right\rfloor + 2 \) indices.

Since successive runs of object pixels must have at least one background
pixel between them, it is clear that there cannot be a direct connection between
different runs of pixels on the same row. The only direct connections that can be
made are between runs on adjacent rows, where the runs overlap (or there is a
diagonal connection for 8-connectivity). With a raster scanned 2×2 window, any
such overlap will continue until the end of the run on either the previous row or
the current row, which provides a condition for their unique detection. This is the
condition for the Union operation, which is used to link the nodes representing
the connected indices. The linking always maintains the connectivity by adding
the node on the current row into the graph, and maintaining a link from the
node on the previous row to the added node.

After the end of a run on the previous row, the associated node can never be
accessed again (although with index recycling, the same index can later be used
for another run). This invokes the Retire operation, which removes the node
from the network. Just as nodes are created in a sequential order, they are also
retired in the exact same order, and the associated links disappear as well.

Since arcs are always from an earlier node to a later node, any node being
retired can have no incoming links, because any such links would have to come
from earlier nodes, which would have already been retired. Removing a retired
node with only one outgoing link will not affect the connectivity of the graph.
A node with two outgoing links may connect two sub-graphs. Therefore, to
maintain connectivity it is necessary to link these, which is achieved by adding
a link from the earlier node to the later node. At any one time the remaining
graph of connected nodes correspond to all of the runs of pixels that can currently
extend the object further. Finally, a node being retired with no outgoing links
has no further connections within the image, and is therefore the last node of a
completed object.

The second aspect is that the final calculated feature vector represents the
complete connected component. Each pixel within a run contributes to the run’s
feature vector as it is being scanned. When a run is retired, its feature vector
(plus any others that have been merged into it) is accumulated into that of
another run within the component, so the data is not lost when the node is
Retired. Therefore, each run will contribute to the final feature vector. When
the final node of the connected component is Retired, its accumulated feature
vector will therefore have contributions from each pixel within the connected
component.
4.2 Validation

To validate the Union-Retire CCA, the output feature vectors were compared with those produced by using a conventional two-pass CCL and analysis algorithm [24]. The algorithms were compared on over 10,000 randomly generated binary images, plus also the approximately 300 images from the USC-SIPI image database [22], binarised using a global threshold using Otsu’s algorithm [14].

All of the images gave identical sets of feature vectors from both algorithms. While this does not guarantee correctness, this regression test provides validation of the algorithm.

4.3 Memory Requirements

Within an embedded system, two factors of primary importance are the memory requirements and computational efficiency. The memory requirements of the proposed Union-Retire algorithm are compared with several state-of-the-art single-pass CCA algorithms in Table 1. Note that the memory requirements for the feature vector depend on the particular features being extracted. However, these are the same as for all of the compared algorithms (those which require two copies of the data table, e.g. [12,25], or those that had large tables because they deferred processing until the end of the frame, e.g. [2,13,17,20], were not considered) so are not included in the table.

| Algorithm               | Row buffer         | U-F Memory                      | Complexity     |
|-------------------------|---------------------|--------------------------------|----------------|
| Double-lookup, [11]     | Label               | Augmented label + chain stack + recycle queue | Linear +20% overhead |
| Zig-zag scan, [4]       | Label +1 bpp zig-zag| Augmented label + recycle queue | Linear         |
| Linked-list based, [18] | 2 bpp               | 3 link tables                  | Linear         |
| Union-Retire            | 1 bpp               | 2 link tables                  | Linear         |

There are two areas where memory is required (assuming a streamed image input). The first is the row buffer required to cache labels for the previous row of the scan window; this is of the width of the image. The second is for the Union-Find (or equivalent) data structures required to manage the connectivity, which have size proportional to the number of labels.

Both the double-lookup method of Klaiber et al. [11] and the zig-zag scan method [4] propagate the labels. Therefore, the row buffer needs to hold the labels assigned in the current row. In addition to this, the zig-zag scan method
requires an additional 1 bit per pixel buffer for zig-zag reordering. In contrast, the linked-list method of Tang et al. [18] and the proposed Union-Retire only need to buffer the original binary image because the labels or indices are assigned deterministically. Therefore the latter two methods require significantly less memory for the row buffer.

The linked-list method assumes 4-connectivity. Tang et al. proposed a filter to convert an 8-connected image to a 4-connected one, and this requires an extra bit per pixel to be buffered to correctly calculate the feature vector. Since the proposed method can be applied with either 4- or 8-connectivity, for 8-connected images, the proposed method uses half of the row buffer memory.

Both the double-lookup and zig-zag scan store augmented labels in their merger table. These comprise the label, and the row number in which the label was first allocated. Both methods also require a recycle queue for label allocation and recycling. In addition, the double-lookup method requires a stack for efficiently implementing path compression. The linked-list method [18] requires 3 tables to represent the links between labels, whereas Union-Retire only requires 2 tables for representing connectivity. Therefore, the proposed Union-Retire approach requires 33% less memory for the Union-Find data structures than the previous best design.

4.4 Computational Complexity

Although the algorithm is quite simple, the computational complexity is not as simple to determine. Feature vector accumulation takes constant processing time per pixel, as accumulating the initial feature vector for each run takes one operation per pixel. As each run is Retired, a single operation is used to accumulate the data into the remaining graph.

The complicating factor is the linking, which is required for both Union and Retire operations. If the node being linked already has two arcs from it, the link is passed onto the next node. A single such deferral is constant time per operation, implying linear computational complexity with the number of pixels. However, if the source node after deferral also has two arcs already, then the computational complexity potentially becomes non-linear.

A formal proof that subsequent deferred links never occur is very complex (see for example [11]). Therefore we have focussed on the analysis of a subset of images, rather than worst case analysis. In processing 10,000+ random images, and the 300 natural SIPI images, a second deferral never occurred. Therefore, for this domain of images, there is at most one deferred link per run of pixels, implying linear computational complexity. If a second deferral ever occurs, it is likely to be very rare, and does not substantially alter this conclusion.

5 Summary and Conclusion

We have proposed a new approach to CCA, which moves away from the conventional Union-Find algorithm. Instead of focussing on the labelling of each
connected component, more attention is given to connectivity, which is represented by a directed acyclic graph. A single pass is made through the image, with runs of consecutive object pixels treated as a single unit. A Union operation links runs connected on adjacent rows by adding an arc to the graph. Whenever a run falls out of the processing window, a Retire operation removes the corresponding node from the graph, and adjusts the links to maintain connectedness. A feature vector of each connected component is built incrementally by accumulating feature vectors of each pixel in a run. When a node is Retired, the associated feature vector is merged with that of a remaining node. When the last node of a connected component is Retired, the associated feature vector is output.

We have shown that this new approach to CCA is correct both through informal proof and regression testing against a standard 2-pass algorithm. Testing has also shown that the algorithm has linear computational complexity, although this is more difficult to prove formally. The Union-FInd memory requirements of the algorithm are shown to be 33% lower than the best state-of-the-art CCA algorithm. These make the Union-Retire approach attractive for embedded image processing.

Future work is to explore a hardware implementation of the Union-Retire algorithm using a field programmable gate array (FPGA).

References

1. Bailey, D.G.: Raster based region growing. In: 6th New Zealand Image Processing Workshop, Lower Hutt, NZ, 29–30 August 1991, pp. 21–26 (1991)
2. Bailey, D., Johnston, C.: Single pass connected components analysis. In: Image and Vision Computing New Zealand, Hamilton, NZ, 5–7 December 2007, pp. 282–287 (2007)
3. Bailey, D.G., Johnston, C.T., Ma, N.: Connected components analysis of streamed images. In: International Conference on Field Programmable Logic and Applications, Heidelberg, Germany, 8–10 September 2008, pp. 679–682 (2008). https://doi.org/10.1109/FPL.2008.4630038
4. Bailey, D.G., Klaiber, M.J.: Zig-zag based single pass connected components analysis. J. Imaging 5(4), article 45, 26 p. (2019). https://doi.org/10.3390/jimaging5040045
5. Bailey, D.G.: History and evolution of single pass connected component analysis. In: 35th International Conference on Image and Vision Computing New Zealand, Wellington, NZ, 25–27 November 2020, pp. 317–322 (2020). https://doi.org/10.1109/IVCNZ51579.2020.9290585
6. Hopcroft, J.E., Ullman, J.D.: Set merging algorithms. SIAM J. Comput. 2(4), 294–303 (1973). https://doi.org/10.1137/0202024
7. Jeong, J., Lee, G., Lee, M., Kim, J.-G.: A single-pass connected component labeler without label merging period. J. Sign. Process. Syst. 84(2), 211–223 (2015). https://doi.org/10.1007/s11265-015-1048-7
8. Johnston, C.T., Bailey, D.G.: FPGA implementation of a single pass connected components algorithm. In: IEEE International Symposium on Electronic Design, Test and Applications, Hong Kong, 23–25 January 2008, pp. 228–231 (2008). https://doi.org/10.1109/Delta.2008.21
9. Klaiber, M.J., Bailey, D.G., Baroud, Y.O., Simon, S.: A resource-efficient hardware architecture for connected component analysis. IEEE Trans. Circuits Syst. Video Technol. 26(7), 1334–1349 (2016). https://doi.org/10.1109/TCSVT.2015.2450371
10. Klaiber, M.J.: A parallel and resource-efficient single lookup connected components analysis architecture for reconfigurable hardware. Ph.D. thesis, Stuttgart University (2017)
11. Klaiber, M.J., Bailey, D.G., Simon, S.: Comparative study and proof of single-pass connected components algorithms. J. Math. Imaging Vis. 61(8), 1112–1134 (2019). https://doi.org/10.1007/s10851-019-00891-2
12. Ma, N., Bailey, D., Johnston, C.: Optimised single pass connected components analysis. In: International Conference on Field Programmable Technology, Taipei, Taiwan, 8–10 December 2008, pp. 185–192 (2008). https://doi.org/10.1109/FPT.2008.4762382
13. Malik, A.W., Thornberg, B., Imran, M., Lawal, N.: Hardware architecture for real-time computation of image component feature descriptors on a FPGA. Int. J. Distrib. Sens. Netw. 10(1), 815378 (2014). https://doi.org/10.1155/2014/815378
14. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. 9(1), 62–66 (1979). https://doi.org/10.1109/TSMC.1979.4310076
15. Sedgewick, R., Wayne, K.: Algorithms, 4th edn. Addison-Wesley Professional, Boston (2011)
16. Selkow, S.M.: One-pass complexity of digital picture properties. J. ACM 19(2), 283–295 (1972). https://doi.org/10.1145/321694.321701
17. Spagnolo, F., Perri, S., Corsonello, P.: An efficient hardware-oriented single-pass approach for connected component analysis. Sensors 19(14), 3055 (2019). https://doi.org/10.3390/s19143055
18. Tang, J.W., Shaikh-Husin, N., Sheikh, U.U., Marsono, M.N.: A linked list run-length-based single-pass connected component analysis for real-time embedded hardware. J. Real-Time Image Process. 15(1), 197–215 (2016). https://doi.org/10.1007/s11554-016-0590-2
19. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM 22(2), 215–225 (1975). https://doi.org/10.1145/321879.321884
20. Thornberg, B., Lawal, N.: Real-time component labelling and feature extraction on FPGA. In: International Symposium on Signals, Circuits and Systems, Iasi, Romania, 9–10 July 2009, pp. 1–4 (2009). https://doi.org/10.1109/ISSCS.2009.5206100
21. Trein, J., Schwarzbacher, A.T., Hoppe, B., Noffz, K.H., Trenschel, T.: Development of a FPGA based real-time blob analysis circuit. In: Irish Signals and Systems Conference, Derry, Northern Ireland, 13–14 September 2007, pp. 121–126 (2007)
22. USC-SIPI: USC-SIPI image database. http://sipi.usc.edu/database/
23. Veillon, F.: One pass computation of morphological and geometrical properties of objects in digital pictures. Sign. Process. 1(3), 175–189 (1979). https://doi.org/10.1016/0165-1684(79)90018-5
24. Wu, K., Otoo, E., Suzuki, K.: Optimizing two-pass connected-component labeling algorithms. Pattern Anal. Appl. 12(2), 117–135 (2009). https://doi.org/10.1007/s10044-008-0109-y
25. Zhao, C., Duan, G., Zheng, N.: A hardware-efficient method for extracting statistic information of connected component. J. Sign. Process. Syst. 88(1), 55–65 (2016). https://doi.org/10.1007/s11265-016-1126-5
26. Zhao, F., Lu, H.Z., Zhang, Z.Y.: Real-time single-pass connected components analysis algorithm. EURASIP J. Image Video Process. 2013:21, 10 (2013). https://doi.org/10.1186/1687-5281-2013-21