Parallel connected-Component-Labeling based on homotopy trees

Fernando Diaz-del-Rio [a], Pablo Sanchez-Cuevas [a], Helena Molina-Abril [b], *, Pedro Real [b]

[a] *Department of Computer Architecture and Technology. University of Seville. Spain*
[b] *Department of Applied Mathematics I. University of Seville. Spain*

Keywords:
Connected-Component-Labeling
Computational topology
Adjacency tree
Digital image
Parallelism

ABSTRACT

Taking advantage of the topological and isotopic properties of binary digital images, we present here a new algorithm for connected component labeling (CLL). A local-to-global treatment of the topological information within the image, allows us to develop an inherent parallel approach. The time complexity order for an image of $m \times n$ pixels, under the assumption that a processing element exists for each pixel, is near $O(log(m+n))$. Additionally, our method computes both the foreground and background CCL, and allows a straightforward computation of topological features like Adjacency Trees. Experiments show that our method obtains better performance metrics than other approaches. Our work aims at generating a new class of labeling algorithms: those centered in fully parallel approaches based on computational topology, thus allowing a perfect concurrent execution in multiple threads and preventing the use of critical sections and atomic instructions.

1. Introduction

Connected Component Labeling (CCL) was one of the first algorithms of topological nature to be developed within the computer vision field (see [7,12,24,27]). There is a plethora of applications of this computational topological process in image analysis, pattern recognition and image understanding. In most of them, CCL is used as a compulsory stage to achieve semantically correct segmentations. By a connected component (CC, for short) of a 2D binary digital image I of $m \times n$ pixels, we mean a set of pixels of the same color (black or white) that are "connected" to each other. Working with pixels regularly placed in a square lattice, the equivalence relationship "to be connected to" is usually understood in terms of paths of consecutive 4-neighbor (sharing an edge) or 8-neighbor (sharing a corner) pixels. Then, CCL is an image processing algorithm that provides a unique label to each connected component of I. Specifying a conventional two-dimensional matrix of 0,1-values as initial image data structure, we can classify 2D binary CCL's algorithms according to their local/global processing and management of the connectivity information into two classes: (a) *iterative algorithms* based on label-propagation techniques and (b) *direct algorithms* based on label-equivalence-resolving.

All of them (including the fastest ones) share the same first step: to scan the whole binary image or its contours pixel-by-pixel. Therefore, they start by labeling the first pixel and so the second one as a function of the first pixel label. This local processing based on neighborhood masks proceeds progressively until the last pixel is reached. This fact necessarily implies data dependencies between the labeling of one pixel and the previous one, which prevents these methods from using a pure parallel approach. In terms of time complexity, this means that linear order $O(m \times n)$ cannot decrease independently of the number of available processing units.

After this provisional labeling, a second step solves the so-called *label equivalence problem*, which constitutes a crucial stage of all modern labeling algorithms. This process can be done: (a) Locally and iteratively by propagation of labels step by step across the image until stabilization or (b) globally and directly by means of an equivalence table holding a graph structure representing the label connections, computing the transitive closure of the graph and, finally, updating label values with the equivalence table (a complete review of the state-of-the-art CCL's algorithms can be found in [8]). An usually employed mechanism to solve the label equivalence problem is the 'union-find' algorithm, that replaces those provisional labels by its representative label, once equivalences (that is, two or more labels that actually belong to the same CC) have been detected. Nevertheless, current union-find proposals are mainly sequential, thus needing critical sections or atomic instructions if many processing elements act in this step concurrently. The maximum number of provisional labels can be as big as $(m \times n/4)$. Hence, the amount of time for this second step may be much bigger than for the first one.

* Corresponding author.
E-mail address: habril@us.es (H. Molina-Abril).

Using duality properties of digital 2*D* topology, a direct CCL algorithm called *CCL based on homotopy trees* (or HT_{CCL}) is designed and implemented here. The main idea is to develop a simplified and refined version in terms of (exclusively) pixels, of the interpixel algorithm described in [20,21]. This method for labeling images is based on a topologically consistent graph model of a 2*D* binary digital image, called Homological Spanning Forest (*HSF*, for short, see [14]). Considering a 2*D* binary digital image as an abstract cell complex ([13]), an HSF structure consists of two non-intersecting trees: one spanning all the cells of dimension 0 (pixels) and the other spanning all the cells of dimension 2 (pixel's corners). A partition of the complete set of 1-dimensional cells is made, in such a way that some of them are nodes of the first tree, and the rest, of the second one. The contractibility of *I* in this reduced topological canvas is maintained, and we benefit from this property to deduce our conclusions. The advantages of HT_{CCL} are:

- It is a fully parallel procedure for labeling CCs. In our parallel framework we can define one processing element (*PE*) per pixel, and all subsequent operations can be executed theoretically at the same time by all the *PEs*. Real computer restrictions must be carefully considered so that the real implemented parallelism approaches the theoretical one. The total amount of memory and mean number of accesses per pixel must be carefully weighed up, as it is the most important parameter in CCL execution (see [5]).

- It is one of the fastest CCL algorithms. Its theoretical time complexity is near the logarithm of the width plus height of the image. Besides, experimentally compared to other implementations of the CCL algorithms ([2,4]), HT_{CLL} presents better metrics. Additionally, our method computes both the foreground and background CCL, thus enriching image processing with background components, and allowing a straightforward computation of topological features like Adjacency Trees (see last item in this list).

- Our work aims at generating a new class of labeling algorithms: those centered in fully parallel approaches based on computational topology, thus allowing a perfect concurrent execution in multiple threads and preventing the use of critical sections and atomic instructions.

- An appropriate modification of HT_{CCL} leads to obtaining additional topological, geometric, analytical and statistical features of the different CC's of the image, giving rise to a complete CCA (Connected Component Analysis) of the image.

- The result of HT_{CCL} might be understood as an useful and efficient compressed topological model for recognition tasks and other high level computer vision applications. The classical Adjacency Tree (AdjT for short, also called topological, inclusion or homotopy tree, see [23,25]) is an example of such kind of representations, which can be automatically obtained from HT_{CCL}, due to the fact that both foreground (FG) and background (BG) components are computed at the same time.

The paper is organized as follows: Section 2 contains a summary of related works. Section 3 introduces the proposed method. Section 4 focuses on the main part of the algorithm, based on the concept of "Transports". Section 5 clarifies some implementation and time complexity details, and in Section 6 our experimental results are shown. The paper ends in Section 7 with some conclusions and future work.

2. Related works

As previously mentioned, topological magnitudes can be computed via two main approaches: iterative and direct (or a combination of both). However, the vast majority of CCL algorithms are mainly based on raster scan, following the first era of sequential computation, as in [23,24].

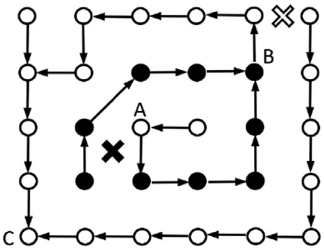
A classical data partition technique for obtaining parallelism consists of dividing the image into strips. The counterpart of this division is that it generates more provisional labels (appearing at the interfaces between each two strips), thus augmenting the time needed for the second stage. Even if the second stage uses a sophisticated union-find technique for the provisional labels, there is also some room for parallelism in it, and many works have addressed different variations (see [6,9,11,16]) including tuning parallel algorithms for specific computers (see [1]). However, the great majority of CCL methods do not take advantage of important topological (duality, homology, ...) and isotopic properties (characteristic of objects embedded in 2D digital ambiance), which could maximize the degree of parallelization in managing local/global and object/ambiance connectivity information.

Computational topology is the ideal mathematical scenario for promoting parallelism in a natural way. The nature of topological information is essentially qualitative, having the additional advantage that its magnitudes are robust under deformations, translations and rotations. Nevertheless, the results in the literature in that sense are rare. Up to now, the only topological invariant that has been calculated using a fully parallel computation is the Euler number (see [3]). Other authors have recently proposed some other parallel algorithms that compute some aspects of the homological properties of binary images ([15]). In addition, some software libraries of flexible C++ (RedHom, [18]) have been developed for the efficient computation of the algebraic homology of sets. These libraries implements algorithms based on geometric and algebraic reduction methods.

In [20], a digital framework for parallel topological computation of 2D binary digital images based on a interpixel scenario was developed, modeling the image as a special abstract cell complex (see [13] for a clarification of this concept). Following this theoretical framework, CCL methods have been designed and implemented in [19,21]. In relation to these previous works, we adapt here our theoretical framework on the basis of the two following basic topological relationships: "being adjacent to" and "being surrounded by". We propose a new and faster design and implementation of the CCL algorithm based on these ideas (code available in [22]). Our simplification allows us to reduce the number of operations for computing the global AdjT, reducing computation time and memory consumption and maintaining the degree of parallelism to every single pixel.

3. The CCL algorithm based on homotopy tree

Correctness of the proposed HT_{CCL} is based on the fact that 8-adjacency is considered for FG pixels and 4-adjacency for BG pixels ((8,4)-adjacency pair). In this way, we take advantage of the powerful duality and isotopic properties that the topological invariants of connected components and holes have in the context of 2D binary digital images based on square pixels. These properties arise from the fact that this (8,4)-adjacency pair allows the image's cell complex to admit a partition onto cell complexes (of black and white pixels) fully embedded in $\mathbb{R}^2$. Note that for 2*D* binary digital images, there are holes of two dimensions: (a) 0-holes that can be seen as FG 8-CCs or BG 4-CCs in which we distinguish a pixel as their representative element; (b) 1-holes that can be seen as digital closed curves at inter-pixel level that constitute the border of a given CC. We select two adjacent pixels of this curve, to represent 1-holes. From now on, the representative pixel of each connected components is labelled (in the Figures) using capital letters. 1-holes, will be pictured using white X symbols for 8-connected holes, and black X symbols for 4-connected ones. These two concepts can be reduced to a single one within the context of binary images since a hole can be seen as a CC that is surrounded by another CC of different color. Note that due to duality, each connected component will have its corresponding hole in the opposite color (except for the outer components of the image border).



-6	-6	-1	-1	-1	-6
-6	-1	1	1	0	-6
-6	7	0	-1	6	-6
-6	6	1	1	6	-6
0	-1	-1	-1	-1	-1

-24	-25	-26	-27	-28	-29
-18	-19	2	1	0	-23
-12	9	0	-1	6	-17
-6	15	14	13	12	-11
0	-1	-2	-3	-4	-5

0	0	0	0	0	0
0	0	22	22	22	0
0	22	14	14	22	0
0	22	22	22	22	0
0	0	0	0	0	0

Fig. 1. Up, left: A binary image pixel tree $\widetilde{HSF}$ where black pixel arrows (and distances) take the N-NE-E direction by default, whereas white ones the S-W direction. Up, right: Local distance matrix Bottom, left: Global jump distance matrix. Bottom, right: CCL of the image.

Roughly speaking, HT_{CLL} is based on the idea of the computation of a global Adjacency or Homotopy Tree at CC's level starting from a local Homotopy Graph at pixel's level.

To get the big picture, let us start with a simple example of the proposed method, applied to the $5x6$ binary digital image I of Fig. 1 (up, left). The different steps of the proposed HT_{CCL} (Fig. 3) are briefly explained for this example. Given a binary digital $2D$ image, a rooted tree that connects all the pixels can be built using a particular direction for each color (those cases leading to graphs are to be explained in the next section). Specifically for this image, each CC is connected by just one arrow to the CC it is surrounded by. More concretely, the following rules have been established in our method for the construction of such graph: For white pixels: first South, then West (in short S-W); for black pixels, first North, then Northeast; then East (in short N-NE-E). Section 5 explains in detail the reasons (related to computing performance and parallelism) for building the tree of black pixels following the opposite direction as for white pixels. Therefore, the first step ($J_{init}()$ of Fig. 3) consists of the generation of a directed spanning graph $\widetilde{HSF}(I)$ over the pixels of the given image I, following certain direction criteria. This concept appears as an important part on the HSF topologically consistent graph model of a $2D$ binary digital image (see [14]).

The $\widetilde{HSF}(I)$ graph, can be directly represented by a matrix of specific jump distances. In fact, the process J_{init} returns this matrix (see Fig. 1 (up, right)), that will be called *Local Jump Matrix* ($LJM(I)$, for short), and is computed as follows: For each pixel, the local distances (those to the pixel that it is connected to) are computed. These distances depend on the chosen adjacency and direction criterion; in our example in Fig. 1, BG local distances can be defined as: -1 (west connection) and -6 (south connection) (because each row counts for a distance of 6, which is the number of columns). Similarly, local distances for FG pixels result to be: $+6$ (north connection), $+7$ (N-E connection), $+1$ (East connection). At the same time, we can mark with a 0 those nodes that connect two different colors. These are to be called "attractors", that are the roots of the subtree of each CC. In other words, a FG attractor is a FG pixel vector-connected (that is, by a single arrow) to a BG pixel in $\widetilde{HSF}(I)$. The definition of a BG attractor is completely analogous.

Then, using attractors as a reference, we can compute global distances to them, and therefore obtaining a matrix of global jump distances, that is the *Global Jump Matrix* $GJM(I)$, in which each value corresponds to the distance of a given pixel to its farthest CC attractor (see Fig. 1, (bottom, left)). The transformation of $LJM(I)$ into $GJM(I)$ corresponds to the second step of Fig. 3, $J_{computation}()$. Let us emphasize that the final CCs of the image have not yet been computed at this stage.

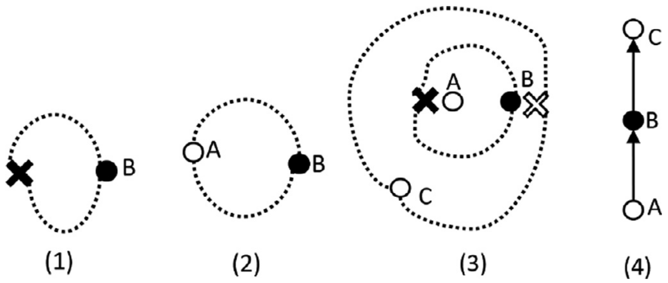


Fig. 2. (1),(2) and (3) Graphical interpretations of the $GJM(I)$ matrix corresponding to Fig. 1 in terms of topological equivalences. (4) Graphical representation of the AdjT model of the image I shown in Fig. 1 (up, left).

A third phase (called *Transports()* in Fig. 3) implies some changes on the $GJM(I)$ matrix, which are necessary when cycles appear after using the previous direction criterion. This is the crucial step of the method, and it is focused on erasing false FG and BG attractors, by pairing them following certain criteria (see next section for further details).

Finally, and using the modified $GJM(I)$ matrix, this process leads to an AdjT and therefore to a correct CCL of the initial image (*Labelling*() in Fig. 3). Then, the final $GJM(I)$ matrix represents a tree, called Connected Component Labeling Tree (*CCLT* for short).

Additionally, further CCA can be directly extracted from the output of the *Transport ()* stage (*OtherFunctions*() in Fig. 3). The resulting CCL of the image in Fig. 1 (up, left) is shown in Fig. 1 (bottom, right). The AdjT can be straightforwardly extracted as the connections among white and black color attractors when following the modified $GJM(I)$ matrix. Going back to the duality concepts introduced at the beginning of this section, this matrix is interpreted in terms of holes, and thus obtaining the corresponding AdjT of the image. Fig. 2 (1) shows the topological equivalence between a black 0-hole (FG attractor labelled as B) and the corresponding black 1-hole (black cross) of the same 8-connected CC that obviously surrounds a white CC. This fact is pictured using a double dotted path. Fig. 2 (2) shows the correspondence between the black 1-hole (paired with the CC labelled B) and the white CCs it surrounds. This connection can be automatically detected within the $GJM(I)$. Fig. 2 (3) pictures all the topological pairings that can be translated to the AdjT model in Fig. 2 (4).

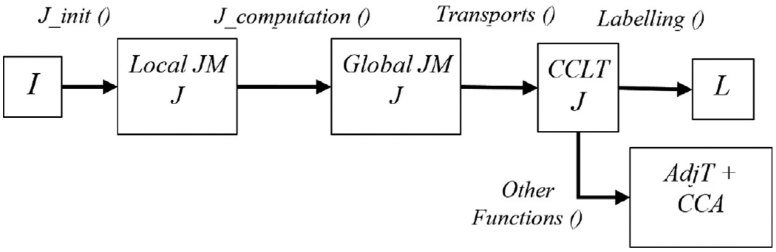


Fig. 3. Phases involved in the parallel CCLT building, labelling and other possible output results (i.e. AdjT and CC geometric information).

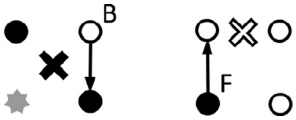


Fig. 4. The two unique possible patterns for attractors. Left: BG attractor (named *B*) that is connected to a 8-adj FG set of pixels. Right: FG attractor *F* connected to a 4-adj BG set of pixels. Crosses represent 1-holes. Grey star represents a pixel of any color.

The key point here is the complete parallelization of this whole process, that will be explained in the following sections. For the rest of the paper let us suppose a binary image $I(x, y)$ having m rows and n columns. Rows (resp. columns) of the matrix associated to I are numbered from top-to-bottom (resp. from left-to-right), starting from 0. A pixel can be identified by a pair of integers (x, y), with x being the column number and y being the row number. The adjacency types can be defined according to pixel coordinates. For pixel $P=(x,y)$, pixels with coordinates $N(P)=(x,y-1)$, $E(P)=(x-1,y)$, $S(P)=(x,y+1)$, $W(P)=(x+1,y)$ are its 4-adjacent; these four in addition with the four pixels $NE(P)=(x-1,y-1)$, $SE(P)=(x-1,y+1)$, $NW(P)=(x+1,y-1)$, $SW(P)=(x+1,y+1)$ complete the rest of its 8-adjacent pixels.

4. Attractor's cancellation for building the CCLT via Transports

It is worth to mention that all the processes and data structures (including the concept of $\widetilde{HSF}$ graph) of our method are only represented by two matrices: the original binary I and the jump matrix J. Using these two matrices, many other structures can be straightforwardly obtained (labelling, AdjT matrix, etc.) as shown in the following. Due to the fact that the two first steps $J_{init}()$ and $J_{computation}()$ are simply based on matrix distance calculus among pixels, we consider that no further explanation is needed. Therefore, we focus here on the *Transport()* stage. This process is based on the "crack transport" method defined in [20]. This crack transport technique consists of interchanging connectivity information between two trees, by moving edges between them. This interchange, is only possible if certain conditions are satisfied. In this previous work, the initial image was represented as a cell complex (see [13] for a complete description of this concept) and two different HSF trees were used.

In the simplified scenario of HT_{CCL}, in which the topological canvas is reduced to a single graph $\widetilde{HSF}$, these conditions can be enunciated in terms of attractors. After building the $GJM(I)$ matrix, and due to the concrete directions taken when building the initial $\widetilde{HSF}$, all the attractors follow one of the two very specific patterns, shown in Fig. 4. Note that the crosses in the Figure representing 1-holes, are not intrinsically implemented within the matrix data structure, but they are shown in order to emphasize the natural pairing (due to duality) among opposite color 0-holes (attractor) and 1-holes.

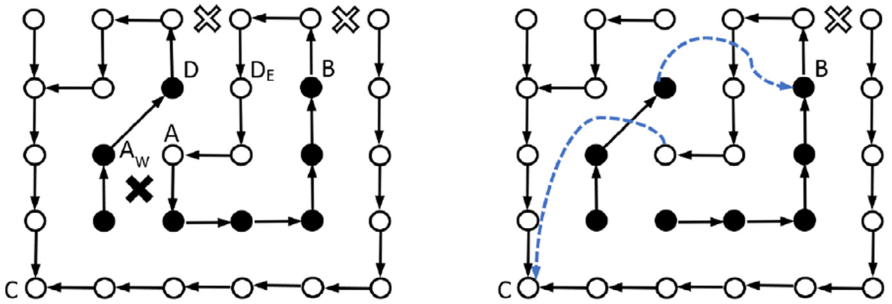
With previous considerations, a false FG attractor can be defined as a pixel whose north and east adjacent BG pixels are connected through $\widetilde{HSF}$ to two different BG attractors. Likewise, a false BG attractor is a pixel whose south, southwest and west adjacent FG pixels are connected to two different FG attractors. Hence, the edges in the $\widetilde{HSF}$ graph are transported, in such a way that a pair of false FG and BG attractors disappears (according to the following Transport method). Any transport implies the updating of two jump matrix elements (or equivalently, redirecting two edges for each pair of false attractors). An example of these changes are shown in Fig. 5. Note that finally the remaining BG attractor is located on the SW corner of the image. However, a black attractor (representative of a false white hole) must be cancelled with only one of the white attractors (which means that the remaining white attractors must find another black attractor). Thus, in the general case we cannot ensure that cancellation of false attractor pairs can be done in parallel. As it is depicted in Fig. 7 (Right), several white attractors (likewise for black attractors) may select the same black attractor (white attractor, resp.) to try the cancellation. Hence, we need a stronger condition (specified in the next Transport method) to cancel false attractors in parallel. This is the key-point of our approach and justifies building tree of black pixels in the opposite direction to that of white pixels.

Transport method to remove false attractors.

We define this method for the FG attractor, and it is equivalent for BG attractors. For every FG attractor and in parallel, we check two possible conditions (in an orderly fashion). For clarity purposes, the process is explained using the notation of Fig. 6.

- For each FG attractor (for instance, F), we select the adjacent East BG pixel (F'_E) and follow the BG tree (downward dashed line), which goes to some BG attractor (F'_{ES}). Then the West adjacent FG pixel F_{ES}, is chosen and the FG tree is followed until a new FG attractor is reached (F). Only if this last FG attractor were the same as the initial one, the transport would be done. Doing a transport implies removing a pair of false (FG and BG) attractors: Those edges that previously connected false attractors with pixels of different color, are moved now so that they end up connecting these false attractors with another attractor of its same color. In Fig. 6, F must be connected now to the black attractor G (that can be found simply by following the tree of G_{NW}, which is at the south of F'_{ES}). Likewise, for the transport of the false BG attractor F'_{ES}, which must finally be connected to F'_{NW}. These new edges are drawn with discontinuous arcs.
- For each FG attractor (for instance, F), we select the adjacent North BG pixel (F'_N) and follow the BG tree (downward dashed line), which goes to some BG attractor (F'_{NW}). Then the South adjacent FG pixel F_{NW}, is chosen and the FG tree is followed until a new FG attractor is reached (F). Only if this last FG attractor were the same as the initial one, the transport would be done (similarly to the previously mentioned one).

Perfect concurrency in executing transports is guaranteed since for each transport there are two directed paths up to their cor-



-24	-25	-26	-13	-28	-29
-18	-19	***0***	-7	0	-23
-12	9	***0***	-1	6	-17
-6	15	14	13	12	-11
0	-1	-2	-3	-4	-5

-24	-25	-26	-13	-28	-29
-18	-19	***2***	-7	0	-23
-12	9	***-14***	-1	6	-17
-6	15	14	13	12	-11
0	-1	-2	-3	-4	-5

Fig. 5. Left: A simple image containing a pair of false attractors *A, D* and its jump matrix. The two 0*s* in italic bold are the attractors to be cancelled when the cycle *D*, D_E, *A*, A_W is detected (see the Transport method). Right: The result after the transport (where changes are indicated by dashed arcs), and its corresponding jump matrix. Note that the two 0*s* have been changed for new jump distances that reveal where the false attractors *A, D* point now (that is, to *B* and *C* resp.). Bottom: jump matrix before and after the transport (changes are highlighted in bold).

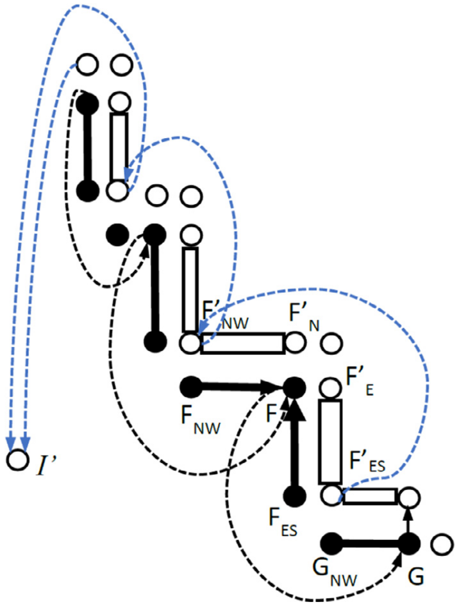


Fig. 6. Left: A black stair like shape with several FG attractors (upward arrows) and BG attractors (downward arrows). Relevant pixels for the transport method are drawn with circles. Right: Resulting pointers after the complete cancellation (using the East white pixel for each false FG attractor) are drawn with discontinuous arcs.

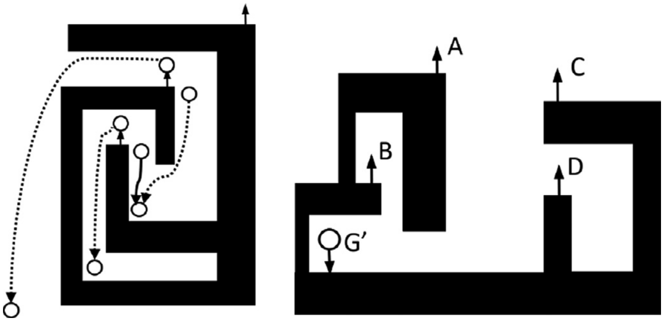


Fig. 7. Left: A spiral like shape where cancellation of false FG/BG attractors cannot be done in just one iteration. Right: Example of black attractors selecting the same white one.

responding roots (attractors), through the sub-trees in the *GJM*(*I*). The condition that the starting pixel must be the same as the destination after in both paths ensures the unicity of the pair to be cancelled. Hence, transport phase has no need for any critical section or atomic clause (see [10]).

For more complex shapes we will not be able to cancel all the false attractors as straightforwardly as in the simple stair-like image of Fig. 6. For instance, this happens for the shape in Fig. 7 (left), where the cancellation of false FG/BG attractors cannot be done in just one iteration. Here, the only possible transport using the BG tree is marked with a continuous arrow, and the rest of dotted arrows does not fulfill the transport requirements. Once the first set of transports is done, we can proceed with a new set to cancel other FG/BG attractor pairs.

This transport phase transforms the jump matrix so as to represent the correct CCLT. It is equivalent to fusing those parts of a same CC, and somewhat similar to the classic 'Union-Find' procedure but benefiting from parallel computation, when using both the BG and FG trees. Obviously, the cancellation of some pairs of false attractors do not guaranteed the complete deletion of all of them. Nevertheless, experiments with hundreds of images reveal that, after a few number of transport iterations, the returned CCL is correct (it contains the same number of labels than other algorithms like those of [4]). The theoretical demonstration of this fact is left for future work.

5. CCLT implementation and time complexity order

The question is now, what parts of a CCLT building can be done in parallel? According to the proposed framework, the parallel CCLT building can be divided into four main phases (Fig. 3). We refer the reader to [19] for a detailed matrix-based description of this process. The time degree order supposing *p PEs* was also fully illustrated in this work.

1) $J_{init}()$. The matrix *J*, containing the jump distances to the attractor, is initialized, thus resulting the *LJM*(*I*). For instance, if FG pixels were traveled using the criterion N-NE-E (in order of priority), the distances to be stored at each element of *J* would be (according to previous order): *n*, *n*+1, +1, being *n* the number of columns of *I*. For BG pixels (criterion S-W), the initial distances would be: $-n$, -1. Time complexity of this phase is clearly $O(mn/p)$, due to the fact that each pixel can write its jump dis-

tance independently of the rest, and only based on the values of its adjacent pixels.

2) $J_{computation}()$. Here the total jump distances to the farthest Northeast FG attractor (southwest BG attractor, resp.) must be computed, thus resulting the $GJM(I)$. To promote parallelism, exponentially distance growing accesses can be executed independently for each pixel. As far as we know, the first work that proposed a similar parallel scheme was [26] with the purpose of producing highly efficient Monte Carlo simulations for two and three-dimensional critical Ising models. Further details for applying this phase to binary images can be found in our previous paper [19], where it is shown that this phase can be executed in $log2(m+n-1)$ iterations at most, due to the exponential nature. Their complexity is $O((mn/p)\log(m+n))$, thus being this phase usually the most time consuming.

3) *Transports()*. In this phase pairs of false attractors are cancelled mutually, this means a transport of edges until a unique tree is obtained: the CCLT. This stage must execute several pairs of cycle searching. Because we use FG attractors to detect cycles, directions for each pair must be first South, and secondly West. This crucial third phase requires a deeper explanation (see Fig. 7 (Left)). According to the transport method, each iteration k can be computed fully in parallel for all the remaining attractors, which supposes a complexity of $O(n_{FGatt}(k)/p)$, being n_{FGatt} the number of FG attractors that still remain at iteration k. In [19], it was found that the number of iterations reached a maximum of six pairs even for the most problematic images, which were big random images (until 16 Mpixels) having a 50% of black pixels. Conversely, it was only one for the real images tested (sizes until 2 Mpixels). Although this phase seems to be tricky, if there were more PE than FG attractors, its time complexity is reduced to a few iterations. The worst case scenario of this phase is left for future research. However, shapes designed to stress algorithms that has been proved as a near worst-case scenario for several connected-component labelling, like Fig. 14 of [5], and the spiral image or the Hilbert space-filling curve (both in Fig. 8 of [17]) are solved in just one and two pairs of iterations of this transport phase, resp.

4) *Labelling*() (and other CCA functions in parallel). Once matrix J represents the true CCLT, final label matrix L can be straightforwardly obtained for each pixel, making each pixel look for its true attractor. Hence, this phase can be done in a fully parallel way for every pixel. As the final matrix comprises all the connectivity information of a binary image, other characteristic and geometric measures can be computed at the same time, like AdjT, CC perimeters and areas, etc.

Some details about implementation: A complete (YACCLAB-compliant [4]) C++/OpenMP code was written ([22]). Although previous phases count theoretically for a time complexity order near the $O(log(m+n))$ (under the assumption that a *PE* exists for each pixel), there were many optimizations needed for handling with real machines. The most relevant ones were related to memory management in order to improve cache access locality. For a better understanding of the real code, and because of the high impact on actual performance, a short description follows:

- When building $LJM(I)$, direction for FG CCs was East-then-Southeast-then-South. For BG CCs North-then-West. At the same time, a list of attractors was recorded in this phase, to ease the transport phase (see below).

- Although all *for* loops can be parallelized by simply adding OpenMP directives to row loops, actual implementation contains parallel sections with #pragma omp parallel because a little additional speed-up is achieved. Moreover, when building $GJM(I)$, that is, when computing total jump distances, instead of the theoretical $log2(m+n-1)$ number of iterations, only one iteration was actually done. That is, two row loops were implemented with only one column loop inside. The first incremented row and column indexes (thus promoting West directions for BG pixels), but the second decremented both indexes (promoting East directions for FG pixels). With these two loops, apart from benefiting for a stronger access locality, each jump value inherits the previous pixel jump. For example, if the image had a completely black row, the first pixel jump would contain a value equal to the number of columns, thus pointing directly to the attractor on the most East pixel. The suppression of all the theoretic iterations reduced considerably total time of this stage but supposed that the total jump distance to the attractor was not fully computed. Hence jump distances had to be updated in the next phase.

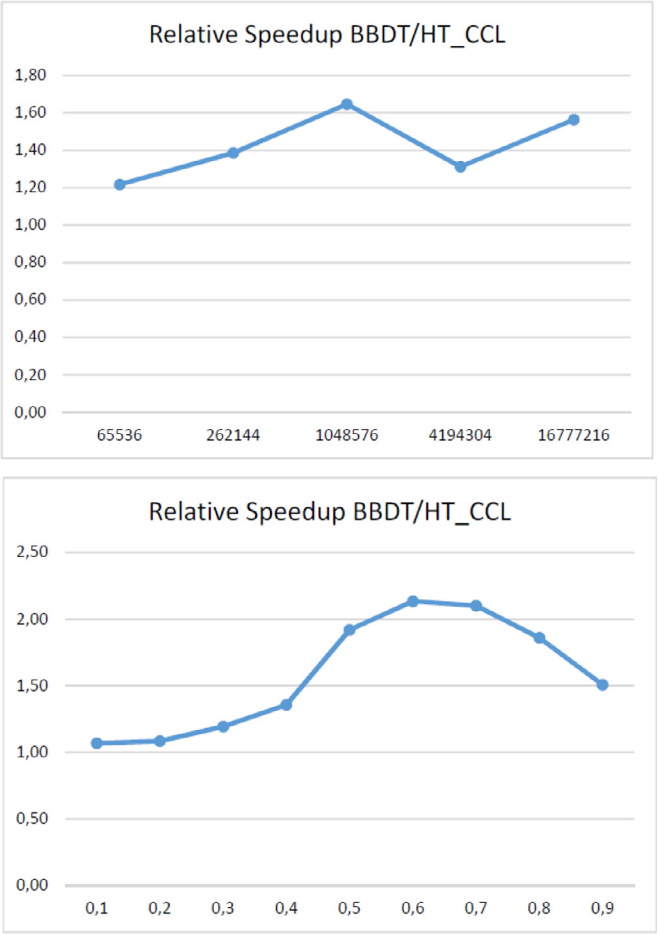


Fig. 8. Mean speedup between *BBDT* and HT_{CCL} for random images of [4]. Up: different sizes (in pixels). Bottom: densities from 0.1 to 0.9.

- Transports were done using the previous recorded list of attractors. During each searching, jump values of the previously canceled false attractors were also updated. This was useful to reduce the number of hops on subsequent transport iterations.

- In the final labelling phase, there were yet some jump distances that had not been plenty updated. Thus, in order to find the definitive attractor of each pixel, some final updating was needed. Additionally, the label assigned to each pixel was the index of its attractor, which is a unique identifier for every CC.

To sum up, the unique CCLT process having real dependences with previous steps is the iterations of the transport stage, and the number of hops needed to find a true attractor. Still, these numbers are not very elevated: it has been empirically shown ([19]) a maximum of four pairs for the first, and maximum of 176 for the second when processing 4Mpixel random images having a 50% of black pixels.

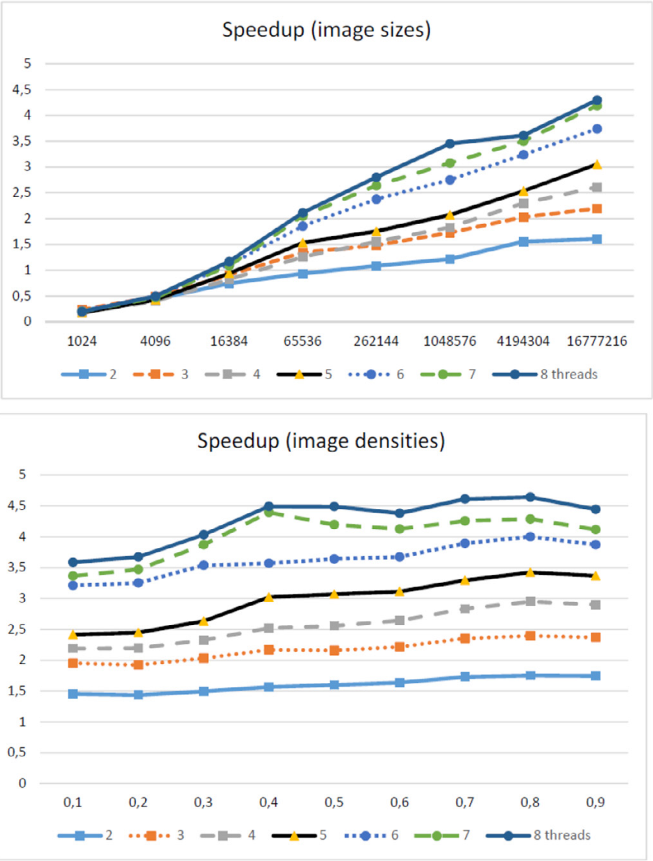


Fig. 9. Up: Mean speedup for 2 to 8 threads when testing the random images of different sizes of [4]. Bottom: Mean speedup for 2 to 8 threads when testing random images of densities from 0.1 to 0.9 (1 thread is the base time).

6. Testing results

Tests were carried out on an Intel Xeon E5 2650 v2 server with: 2.6 GHz, 8 cores, 8x32 KB data caches, Level 2 cache size 8x256 KB, Level 3 cache size 20 MB, maximum RAM bandwidth: 59.7 GB/s. Experiments were run following the procedure of the Project[4] for random images of different sizes (in a nutshell, tests are run 10 times, the minimum execution time for each image was taken, which allows to achieve more reproducible results and reduce delays produced by other running processes). It is important to mention that comparison between a classical CCL algorithms and our method is not fair since our method computes a complete topological representation of the image, whereas classical CCL methods return only black CC labels, which impedes them to extract other topological representations like the AdjT. That is to say, our proposal computes both the foreground and background CCLs, which supposes processing the double quantity of pixels (approximately, this obviously depends on the image black density). Despite this, for multicore processors HT_{CLL} presents better metrics than the fastest CCL algorithm in YACCLAB project (BBDT by [4]). Thus, only this fastest algorithm is compared with ours in the following figures. Fig. 8 (up and bottom) shows timing ratios when using random images with different sizes and densities (percentage of FG pixels), using the collected timings as generated by YACCLAB project. Likewise, Fig. 9 (up and bottom) shows the speedups when launching different number of threads (same images as in previous figure). Speedups (time ratios between one thread and several ones) are satisfactory for all image sizes and densities. Only for little images, the extra overhead time, introduced by OpenMP when creating the threads, hinders speedup. This supposes that speedups are also decreased for medium images. Finally, an additional advantage of HT_{CLL} is that it presents lower deviation for a same size and different densities than the BBDT method. This is manifest when processing images of very different textures (Fig. 8, bottom). Taking into account the satisfactory scalability and the inherent parallelism, we expect for our method to run even faster in other architectures (massive multicore processors, GPUs, etc.).

7. Conclusions and future work

The HT_{CCL} algorithm can be seen as a refined version of the algorithm of [19,20] in the following senses: (a) Instead of considering an HSF-model of I as a two-dimensional set of physical pixels, we work with the simplified HSF-model of I as a two-dimensional puzzle of pieces of four mutually adjacent pixels. Due to the contractibility of I and the duality properties of the (8,4)-adjacency, our topological canvas is reduced here to a single graph $\widetilde{HSF}$; (b) the crack transport technique of [20] is here substantially modified by suitably defining FG and BG attractor patterns and performing in parallel steps its corresponding pairings. The theoretical time complexity of our CCL computation is logarithmic, and performed experiments show that the proposed method obtains performance metrics that improve other approaches. Additionally, our method allows a straightforward computation of the Adjacency Tree topological model, and computes both the foreground and background CCL, thus enriching image processing. Further improvements of the proposed method will be addressed in the future: (a) Extension to nD dimensions, (b) Extension to color images and Region Adjacency Graphs and (c) Inclusion of additional (geometrical, statistical, analytical, topological, etc.) CCA.

Declaration of Competing Interest

None.

Acknowledgements

Work supported by the Spanish projects MTM2016-81030-P (AEI/FEDER,UE) and TEC2012-37868-C04-02 of Ministerio de Economía y Competitividad and the VPPI of the US.

References

[1] P. Bhattacharya, Connected component labeling for binary images on a reconfigurable mesh architecture, Syst. Archit. 42 (4) (1996) 309–313.
[2] F. Bolelli, M. Cancilla, L. Baraldi, C. Grana, Toward reliable experiments on the performance of connected components labeling algorithms, J. Real-Time Image Process.1–16.
[3] F. Chiavetta, V. Di Gesu, Parallel computation of the euler number via connectivity graph, Pattern Recognit. Lett. 14 (1993) 849–859896.
[4] C. Grana, F. Bolelli, L. Baraldi, R. Vezzani, YACCLAB - Yet Another Connected Components Labeling Benchmark, in: 23rd International Conference on Pattern Recognition, 2016.
[5] C. Grana, D. Borghesani, R. Cucchiara, Optimized block-based connected components labeling with decision trees, IEEE Trans. Image Process. 19 (6) (2010) 1596–1609.
[6] S. Gupta, D. Palsetia, M. Patwary, A. Agrawal, A. Choudhary, A new parallel algorithm for two-pass connected component labeling, IEEE IPDP Symposium (2014) 1355–1362.
[7] R. Haralick, Some neighborhood operations. in real-time parallel computing image analysis (1981) 11–35.
[8] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, Y. Chao, The connected-component labeling problem: a review of state-of-the-art algorithms, Pattern Recognit. 70 (1) (2017) 25–43.
[9] A. Hennequin, L. Lacassagne, L. Cabaret, Q. Meunier, A new direct connected component labeling and analysis algorithms for, Gpus (2018) 76–81.
[10] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, 6th Edition, 19, The Morgan Kaufmann Series, 2017.
[11] O. Kalentev, A. Rai, S. Kemnitz, R. Schneider, Connected component labeling on a 2d grid using cuda, J. Parallel Distrib. Comput. 71 (2011) 615–620.
[12] T. Kong, A. Rosenfeld, Topological algorithms for digital image processing, 19, 1996.
[13] V. Kovalevsky, Algorithms in digital geometry based on cellular topology, 10th IWCIA, Springer Berlin Heidelberg 3322 (2004) 366–393.

[14] H. Molina-Abril, P. Real, Homological optimality in discrete morse theory through chain homotopies, Pattern Recognit. Lett. 11 (2012) 1501–1506.

[15] A. Murty, V. Natarajan, S. Vadhiyar, Efficient homology computations on multicore and manycore systems, 20th Annual International Conference on High Performance Computing (2013) 333–342.

[16] M. Patwary, M. Ali, P. Refsnes, F. Manne, Multi-core spanning forest algorithms using the disjoint-set data structure, In 26th IEEE IPDP Symposium (2012) 827–835.

[17] D. Playne, K. Hawick, A new algorithm for parallel connected-component labelling on gpus, IEEE TPDS 29 (6) (2018).

[18] REDHOM 2017.

[19] F. Díaz del Río, H. Molina-Abril, P. Real, Computing the component-labeling and the adjacency tree of a binary digital image in near logarithmic-time, Computational Topology in Image Context, LNCS, Springer 11382 (2019) 82–95.

[20] F. Díaz del Río, P. Real, D. Onchis, A parallel homological spanning forest framework for 2d topological image analysis, Pattern Recognit. Lett. 83 (2016) 49–58.

[21] F. Díaz del Río, P. Real, D. Onchis, Labeling color 2d digital images in theoretical near logarithmic time, LNCS 10425 (2017) 391–402.

[22] F. Díaz del Río, P. Sánchez-Cuevas, H. Molina-Abril, P. Real., https://www.mathworks.com/matlabcentral/fileexchange/71597-labeling_ht_ccl.

[23] A. Rosenfeld, A. Kak, Digital Picture Processing, Morgan Kaufman (1982).

[24] A. Rosenfeld, J. Platz, Sequential operator in digital pictures processing, J. ACM 13 (4) (1966) 471–494.

[25] J. Serra, Image Analysis and Mathematical Morphology, Academic Press, Inc., 1982.

[26] R.H. Swendsen, J. Wang, Nonuniversal critical dynamics in monte carlo simulations, Phys. Rev. Lett. 58 (1987) 86–88.

[27] F. Veillon, One pass computation of morphological and geometrical properties of objects in digital pictures, Signal Process. 1 (3) (1979) 175179.