On minimum spanning tree streaming for hierarchical segmentation

Leonardo Gigli, Santiago Velasco-Forero, Beatriz Marcotegui

Version of Record: https://www.sciencedirect.com/science/article/pii/S016786552030252X
Manuscript_5f4e1338103765678ff1c4ceae23b32b

Pattern Recognition Letters
journal homepage: www.elsevier.com

On minimum spanning tree streaming for hierarchical segmentation

Leonardo Gigli[a], Santiago Velasco-Forero[a], Beatriz Marcotegui[a]

[a]*Centre de Morphologie Mathématique - MINES ParisTech - PSL Research University, 35 Rue Saint-Honoré, 77300 Fontainebleau, France*

Article history:

Minimum Spanning Tree, Streaming Processing, Hierarchical Segmentation, Mathematical Morphology

ABSTRACT

The minimum spanning tree (MST) is one the most popular data structure used to extract hierarchical information from images. This work addresses MST construction in streaming for images. First, we focus on the problem of computing a MST of the union of two graphs with a non-empty intersection. Then we show how our solution can be applied to streaming images. The proposed solution relies on the decomposition of the data in two parts. One *stable* that does not change in the future. This can be stocked or used for further treatments. The other *unstable* needs further information before becoming stable. The correctness of proposed algorithm has been proven and confirmed in the case of morphological segmentation of remote sensing images.

1. Introduction

Image analysis involves extracting meaningful information from digital images. Tasks such as locate, characterize and identify objects are very common in this field. Image segmentation algorithms are used for image partition and to isolate objects or regions of interest. Hierarchical segmentation provide much richer information. These algorithms produce a sequence of nested partitions of increasing coarseness. Each one contains the representation of items at different scales. Over the years, several graph-based hierarchical segmentation methods have been developed in Mathematical Morphology Meyer (2001b, 2015); Zanoguera et al. (1999); Cousty et al. (2018). Computing a MST is a fundamental step for many of these algorithms. For quasi-flat-zones Najman et al. (2013); Zanoguera and Meyer (2002) a MST on the image gradient is calculated to achieve a characterization of hierarchical structures. In Cousty et al. (2018) a bijection between saliency maps and hierarchies based on quasi-flat zones is provided. Furthermore, a MST is calculated for other image analysis methods such as in hierarchical segmentation Meyer (2001a), marker-based segmentation (Couprie et al., 2011), prior-based segmentation Fehri et al. (2017), salience detection Tu et al. (2016), superpixel segmentation Wei et al. (2018).

This paper is an extended version of the work published by Gigli et al. (2018). The original paper addresses the problem of computing a MST on image streaming . The main contributions of this version are:

- A new simpler method to produce MST on streaming
- Compare the novel method against the one presented in a previous paper by Gigli et al. (2018).
- Illustrate the streaming extension of three MST-based segmentation algorithms.

In order to validate our method we focus on Remote Sensing (RS) applications. Recent advances in RS and computer techniques give birth to the explosive growth of RS data Ma et al. (2015). The straight application of classical Image Processing (IP) techniques cannot be used for real-time processing applications such as streaming. Classical IP methods needs to wait until all the data is known. Furthermore, in many cases the images are too big to fit entirely in memory. Therefore, it is necessary to split the images in strips or tiles before treating them Matas et al. (2008); Gazagnes and Wilkinson (2019). Accordingly, streaming spatial algorithms have been identified as one of the main research challenges in RS Li et al. (2016).

The rest of the paper is organized as follows: Section 2 introduces the problem and notation. Successively, Section 3 presents two algorithms to solve our problem and proves their

e-mail: name.surname@mines-paristech.fr (Leonardo Gigli)

correctness. Section 4 reports quantitative evaluation of our methods. Furthermore Section 5 proposes applications of our method to hierarchical morphological segmentation. Finally, we draw the conclusions in Section 6.

2. Problem and methodology

Let us start introducing the context and the mathematical elements we are going to deal with. Given an image I, we can associate to it a 4-connected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, with nodes $\mathcal{V} = \{p_1, \ldots, p_n\}$ being all image pixels and edges $\mathcal{E} = \{e_1, \ldots, e_m\}$ between neighboring pixels, and weighting edges by color/intensity differences. For example, if we assume I be a grayscale image we can define $\mathcal{W}(e) = |I(p) - I(q)|$, where $e = (p, q)$ and $I(p)$ is the image intensity at pixel p.

Briefly a MST of a graph $\mathcal{G}$ is a subgraph $\mathcal{T}$ that spans all the nodes of $\mathcal{G}$. It contains no cycle and it is such that the sum of the weights of its edges is minimal. Prim and Kruskal are the most well known algorithms (Wu and Chao, 2004) to extract a MST from a graph $\mathcal{G}$. Kruskal method runs in $O(|\mathcal{E}| \log(|\mathcal{E}|))$ time in worst case, where $|\cdot|$ is the function that measures the cardinality of a set. Whilst the implementation of Prim that uses Fibonacci heap runs in $O(|\mathcal{E}| + |\mathcal{V}| \log(|\mathcal{V}|))$. In the case of graph associated to images, (Bao et al., 2014) developed a Prim-based algorithm. The authors propose a method to compute a MST for a 8-bit depth that runs in $O(|\mathcal{V}|)$ time. Furthermore, Prim and Kruskal algorithms have been applied also for clustering identification problems. See (Kim, 2009) for further details. However, in this context the biggest challenge is the processing of huge data volume. To solve this problem, (Olman et al., 2009) proposed a parallel Prim-based algorithm to construct a MST. Similarly to the methods we present later, their method split data in disjoint chunks. Extract a MST for each chunk and a MST for bipartite graphs between two neighboring chunks. Then merge all the MSTs together and finally extract a MST of the complete data set from this union. Unlike the previous case, our methods split the data in non-disjoint chunks. This give us as main advantage, the footprint memory is reduced. The graphs are decomposed between a *stable* and an *unstable* part. The first belongs to the final MST. So, it can be stored and it is not modified anymore. From the second we have to extract the remaining of the final MST. In this way, a streaming application can use data on the fly based on stable regions and the footprint memory is reduced. Therefore bigger data sizes can be processed.

In this paper we focus our attention on streaming applications. Let us introduce the streaming image problem. Consider the simple case of an image I decomposed in two blocks B_1, B_2 and sent one after another. Let B_1 be the first block arriving. Suppose that we compute its MST. The question is, how to compute the MST for the whole image, $I = B_1 \cup B_2$, when B_2 arrives and exploit the information extracted from B_1? Before tackling this problem, basic definitions from graph theory and notation used in this paper will be given.

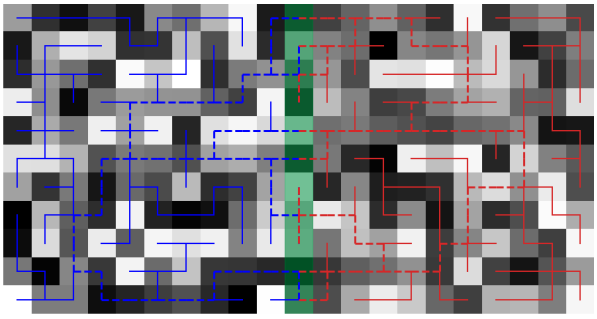


Fig. 1: $\mathcal{T}_0 = \mathcal{MST}(\mathcal{I}_0)$ in red and $\mathcal{T}_1 = \mathcal{MST}(\mathcal{I}_1)$ in blue. $E_{\mathcal{T}_0}$ and $E_{\mathcal{T}_1}$ in bold and dashed, edges linking common pixels (in emerald) and candidate to form cycles on the union of the two MST.

Definition 1 (Graph Union)**.** *Let $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1, \mathcal{W}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2, \mathcal{W}_2)$ two weighted undirected graphs, such that*

$$\mathcal{W}_1\big|_{\mathcal{E}_1 \cap \mathcal{E}_2} \equiv \mathcal{W}_2\big|_{\mathcal{E}_1 \cap \mathcal{E}_2},$$

where $\mathcal{W}\big|_{\mathcal{E}}$ is the restriction of the function $\mathcal{W}$ to the set $\mathcal{E}$. We call $\mathcal{G}_1 \cup \mathcal{G}_2$, the weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ with $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$, and for all $e \in \mathcal{E}$:

$$\mathcal{W}(e) = \begin{cases} \mathcal{W}_1(e) & \text{if } e \in \mathcal{E}_1, \\ \mathcal{W}_2(e) & \text{if } e \in \mathcal{E}_2. \end{cases}$$

Figure 1 illustrates the union of two minimum spanning trees.

Definition 2. *Given a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ and a subset $\mathcal{E}' \subseteq \mathcal{E}$ of the edges, we call $\mathcal{G} - \mathcal{E}'$ the graph $(\mathcal{V}, \mathcal{E} \setminus \mathcal{E}', \mathcal{W})$ obtained by removing the edges $\mathcal{E}'$ from $\mathcal{G}$.*

Furthermore, $\mathcal{MST}(\cdot)$ indicates any function that returns a minimum spanning tree of $\mathcal{G}$. $\mathcal{E}(\mathcal{G})$ is the set of all edges in $\mathcal{G}$. In order to simplify the notation, from now on we treat both images and graphs as the same objects.

Finally, we come back to our question. Let $\mathcal{G}_t$, a graph streaming over time. This is, at each interval t a new block of the complete graph $B_t = (\mathcal{V}_t, \mathcal{E}_t, \mathcal{W}_t)$ arrives. So it holds:

$$\mathcal{G}_t = \bigcup_{s=0}^{t} B_s.$$

Assume that at time $t-1$ the intersection between $\mathcal{G}_{t-1}$ and the new block B_t is known and it is never empty. In this context, we address the problem of updating the MST of the graph $\mathcal{G}_{t-1}$ each time that a block B_t arrives. In particular, we show two algorithms capable to build a MST of the graph $\mathcal{G}_t$, that exploit information coming from time $t-1$.

We conclude this section showing how this formulation can be applied to a stream of images. Let $\mathcal{I}_t$ be an image streaming over time. Without loss of generality, assume that new pixels come from one side of the image, for example the right side of image. If B_t is the new block at time t, for $t = 0, \ldots, T$, we have $\mathcal{I}_t = \mathcal{I}_{t-1} \cup B_t$. The last column of $\mathcal{I}_{t-1}$ is also the first column of B_t, as in Figure 2.

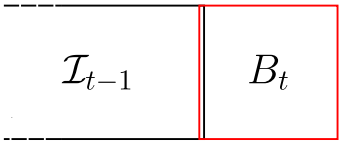


Fig. 2: Streaming of image $\mathcal{I}_t$ as the union of the two non disjoint images $\mathcal{I}_{t-1}$ and B_t. Without loss of generality they share a column of pixels.

For other streaming configurations, as for example a $2D$ tiling, the only difference is that the common pixels should be along the common tiling boundaries.

3. Proposed algorithms

We introduce two methods to compute a MST for streaming by examining the case of the union $\mathcal{G}$ of two MSTs $\mathcal{T}_0$ and $\mathcal{T}_1$, as shown in Figure 1. In order to extract a MST from the graph $\mathcal{G}$ we need to locate all the cycles contained in it. Then, remove the heaviest edges from them. Therefore, as first step, we need to find edges forming cycles in $\mathcal{G}$. Please remark that all cycles contained in the union pass through the pixels in the common intersection between the two trees at least twice. To simplify our explanation, from now on we will refer to this common intersection as frontier. Indeed, it is straightforward to prove that a cycle in $\mathcal{G}$ is generated each time the two trees do not share an edge in the frontier. When this happens, we can find two different simple paths to join two vertices in the frontier. Without loss of generality, let u, v be two vertices in the frontier. Let assume that $e = (u, v) \in \mathcal{T}_0$ but $e \notin \mathcal{T}_1$. Since $\mathcal{T}_1$ is connected we can find a different path $\pi = \{v_0 = u, \dots, v_n = v\}$ in $\mathcal{T}_1$ that joins u and v. Thus $\pi \cup \{e\}$ is a cycle in $\mathcal{G}$. Generalizing this idea, we can retrieve all the cycles looking for the subgraph $\mathcal{G}'$ made by the union of the subgraph $E_{\mathcal{T}_0}$ and $E_{\mathcal{T}_1}$ that contains all the simple paths in $\mathcal{T}_0$ and $\mathcal{T}_1$ that link any two vertices of the frontier. In Figure 1 we draw the edges that belong to $\mathcal{G}' = E_{\mathcal{T}_0} \cup E_{\mathcal{T}_1}$ with bold and dashed lines respectively. Since we are working with trees, it turns out it is easier to find all the simple paths from any node in the frontier N to a special node r marked as root. To do so, we mark one node in the frontier as root of the MST and traverse the tree twice. The first time, we traverse the tree in a top-down fashion using the depth-first-search-algorithm to build the vector of predecessors. The second time, the tree is traversed from bottom-up. This allows us to store all the edges in simple paths from the root to any other node in the frontier. We call this procedure *find_unstable_edges*. See the pseudo-code in Procedure 1. The name of this last procedure will be clearer in the next section. Its main purpose is to identify the edges in the tree that may possibly generate a cycle in the following iterations. From now on, with the term *path* we mean *simple path*.

The two methods that we developed[1] to find a MST of $\mathcal{G}_t$, using information coming from $\mathcal{G}_{t-1}$ are as follows:

- **Streaming Spanning Tree v1:** At the step t, the graph $\mathcal{G}$ made by the union of $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$ may contain cycles. The idea is to identify all the edges that may cause cycles using the Procedure 1 on both graphs $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$. Let $E_{\mathcal{G}_{t-1}}$ and E_{B_t} be the graphs made by all the vertices and edges appearing in all the paths in $\mathcal{MST}(\mathcal{G}_{t-1})$ and $\mathcal{MST}(B_t)$ respectively connecting any two nodes in the frontier N_t. The method computes $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$, and finally it returns:

$$\mathcal{MST}(\mathcal{G}_t) = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \quad \cup (\mathcal{MST}(B_t) - E_{B_t}) \\ \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$$

- **Streaming Spanning Tree v2:** At step t, we consider the graph $\mathcal{G}$ made by the union of $\mathcal{MST}(\mathcal{G}_{t-1})$ and B_t. As in v1, we consider the graph $E_{\mathcal{G}_{t-1}}$ composed of the paths in $\mathcal{MST}(\mathcal{G}_{t-1})$ that link any two vertices of the common frontier because they may possibly generate cycles in $\mathcal{G}$. Instead of computing a MST for the newly arrived B_t. We compute a MST of B_t combined with the candidate edges to form cycles of the previous step. The result is added to $\mathcal{MST}(\mathcal{G}_{t-1})$ without the candidate edges to form cycles.

$$\mathcal{MST}(\mathcal{G}_t) = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$$

Please remark that graph $\mathcal{MST}(\mathcal{G}_{t-1})$ contains edges coming from B_s, $\forall s = 0, \dots, t-1$. Thus the subgraph $E_{\mathcal{G}_{t-1}}$ could contain edges that belong to any previous block. Procedures 2 and 3 shows the pseudo-code for the two methods. In order to prove the correctness of our methods, we prove the following theorem.

Procedure 1 find_unstable_edges

Input: A minimum spanning tree $\mathcal{T}$, root node r and the list of nodes $N = [n_1, \dots, n_h] \subset \mathbb{N}$ in the frontier
Output: E list of edges linking r to another frontier node N.

```
1:  procedure FIND_UNSTABLE_EDGES
2:      // each node in the graph is identified by a number n ∈ ℕ
3:      E ← ∅ // edges to return
4:      V ← (False, ..., False) // visited nodes
5:      // p[n] : predecessor map of tree nodes. p[r] = −1.
6:      p ← depth_first_order(T, r)
7:      for n ∈ N do
8:          while p[n] >= 0 & V[n] == False do
9:              E ← E ∪ {(n, p[n])}
10:             V[n] = True
11:             n ← p[n]
```

Theorem 1. *All the proposed methods return a minimum spanning tree for the graph $\mathcal{G}_t$, for each t.*

Proof. We only prove that the second method returns a MST. For a proof of the first method please refer to our previous work (Gigli et al., 2018). We show that the graph

$$\mathcal{T}_t = (\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$$

returned by the Streaming spanning tree v2 at time t respects the following three properties: 1) $\mathcal{T}_t$ is connected, 2) $\mathcal{T}_t$ is a spanning tree, 3) the sum of all weights $\sum_{e \in \mathcal{E}_t} w_e$, is minimal.

[1] https://github.com/liubigli/SST

Procedure 2 Streaming Spanning Tree v1

Input: A streaming graph $\mathcal{G}_t$
Output: A MST for the graph $\mathcal{G}_t$

procedure STREAMING_SPANNING_TREE_V1
  $\mathcal{T}_0 \leftarrow \mathcal{MST}(\mathcal{G}_0)$
  // N_1 *frontier with block* B_1, $r_1 \in N_1$
  $E_{\mathcal{G}_0} = find\_unstable\_edges(\mathcal{T}_0, r_1, N_1)$
  $\mathbf{F}_0 \leftarrow \mathcal{T}_0 - E_{\mathcal{G}_0}$
  $\mathbf{T}_0 \leftarrow \mathbf{F}_0 \cup E_{\mathcal{G}_0}$
  while a new block B_t arrives **do**:
    $\mathcal{T}_t \leftarrow \mathcal{MST}(B_t)$
    // N_t *frontier with* $\mathcal{T}_t$, $r_t \in N_t$
    $E_{B_t} \leftarrow find\_unstable\_edges(\mathcal{T}_t, r_t, N_t)$
    $\mathcal{F}_t \leftarrow \mathcal{T}_t - E_{B_t}$
    $\mathcal{T} \leftarrow \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup E_{B_t})$
    $\mathbf{T}_t \leftarrow \mathbf{F}_{t-1} \cup \mathcal{F}_t \cup \mathcal{T}$
    // *Fetching* $E_{\mathcal{G}_t}$ *for next iteration*
    // N_{t+1} *frontier with block* B_{t+1}, $r_{t+1} \in N_{t+1}$
    $E_{\mathcal{G}_t} \leftarrow find\_unstable\_edges(\mathbf{T}_t, r_{t+1}, N_{t+1})$
    $\mathbf{F}_t \leftarrow \mathbf{T}_t - E_{\mathcal{G}_t}$

Procedure 3 Streaming Spanning Tree v2

Input: A streaming graph $\mathcal{G}_t$
Output: A minimum spanning tree MST for the graph $\mathcal{G}_t$

procedure STREAMING_SPANNING_TREE_V2
  $\mathcal{T}_0 \leftarrow \mathcal{MST}(B_0)$
  // N_1 *frontier with block* B_1, $r_1 \in N_1$
  $E_{\mathcal{G}_0} = find\_unstable\_edges(\mathcal{T}_0, r_1, N_1)$
  $\mathbf{F}_0 \leftarrow \mathcal{T}_0 - E_{\mathcal{G}_0}$
  $\mathbf{T}_0 \leftarrow \mathbf{F}_0 \cup E_{\mathcal{G}_0}$
  while a new block B_t arrives **do**:
    $\mathbf{T}_t \leftarrow \mathbf{F}_{t-1} \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$
    // *Fetching* $E_{\mathcal{G}_t}$ *for next iteration*
    // N_{t+1} *frontier with block* B_{t+1}, $r_{t+1} \in N_{t+1}$
    $E_{\mathcal{G}_t} \leftarrow find\_unstable\_edges(\mathbf{T}_t, r_{t+1}, N_{t+1})$
    $\mathbf{F}_t \leftarrow \mathbf{T}_t - E_{\mathcal{G}_t}$

First of all, we prove that $\mathcal{T}_t$ is connected.

We show that for any $u, v \in \mathcal{G}_t$ there exists a path $\pi = \{v_0 = u, \ldots, v_n = v\}$ contained in $\mathcal{T}_t$. Let now consider

$$\mathcal{T}'_t = \mathcal{MST}(\mathcal{G}_{t-1}) \cup \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t) \supseteq \mathcal{T}_t.$$

By construction $\mathcal{T}'_t$ is connected since is the union of two connected graphs with a non empty intersection. For this reason, it is possible to find a path $\pi = \{v_0 = u, \ldots, v_n = v\}$ contained in $\mathcal{T}'_t$. Using the fact that $\mathcal{T}'_t \supseteq \mathcal{T}_t$, we can conclude that either $\pi \subseteq \mathcal{T}_t$ or it must exists an edge $e = (v_i, v_{i+1})$ such that $e \notin \mathcal{E}(\mathcal{T}_t)$. In this last case, by construction $e \in E_{\mathcal{G}_{t-1}}$, but $e \notin \mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$. However, since $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ is a connected tree we can find another path $\pi_1 = \{w_0 = v_i, \ldots, w_m = v_j\} \subseteq \mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ and thus in $\mathcal{T}_t$. Repeating this procedure for all the edges $e = (v_i, v_{i+1})$ of the path π not in $\mathcal{E}(\mathcal{T}_t)$, we can build a path π' from u to v entirely contained in $\mathcal{T}_t$. Therefore $\mathcal{T}_t$ is connected.

We prove that $\mathcal{T}_t$ is a tree by contradiction. Let's assume that

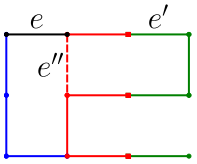


Fig. 3: In black the edge e, while in blue the edges in $\mathcal{MST}(G_{t-1} - E_{\mathcal{G}_{t-1}})$, in red edges coming from $E_{\mathcal{G}_{t-1}}$. In particular, the dashed red edges are edges initially in $E_{\mathcal{G}_{t-1}}$ but not in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$.

$\mathcal{T}_t$ contains cycles. Thus, suppose that $\exists v \in \mathcal{V}(\mathcal{G}_t)$ and a path $\pi = \{v_0 = v, \ldots, v_n = v\}$ that is a cycle in $\mathcal{T}_t$.

By definition of $\mathcal{T}_t$ it is straightforward that such a cycle cannot be contained entirely in $\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$ nor in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, since they are respectively a forest and a tree. So the cycle π must pass through the nodes and edges of both graphs. In particular it must cross at least twice the frontier between $\mathcal{G}_{t-1}$ and B_t. By definition, all paths in $\mathcal{G}_{t-1}$ linking two nodes of the frontier are included in $E_{\mathcal{G}_{t-1}}$. Therefore $\mathcal{MST}(\mathcal{G}_{t-1}) - E_{\mathcal{G}_{t-1}}$ can not contain such edges. As a consequence $\mathcal{T}_t$ has no cycles.

Finally, we prove the third property by contradiction. We define the cost of a graph $\mathcal{G}$ as the sum of its weights $cost(\mathcal{G}) = \sum_{e \in \mathcal{E}(\mathcal{G})} w_e$. So, let suppose that $\mathcal{T}_t$ is not minimal. Then there exists $e \in \mathcal{G}_t$ s.t. $\mathcal{T}_t \cup \{e\}$ contains a spanning tree $\mathcal{T}$ such that $cost(\mathcal{T}) < cost(\mathcal{T}_t)$. Remark that by definition of $\mathcal{T}_t$, the edge e cannot belong to $\mathcal{E}(E_{\mathcal{G}_{t-1}} \cup B_t)$. Otherwise the edge e would be contained also in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, which is a contradiction. Let now consider e' the edge in $\mathcal{T}_t$ replaced by e in $\mathcal{T}$. Since the two trees differ only by the two edges it holds $w_e < w_{e'}$. Two cases are possible:

i) e' is an edge originally in $\mathcal{MST}(\mathcal{G}_{t-1})$,
ii) e' is an edge originally in the new block B_t, this is, $e' \in \mathcal{E}(\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)) \cap \mathcal{E}(B_t)$, as in figure 3.

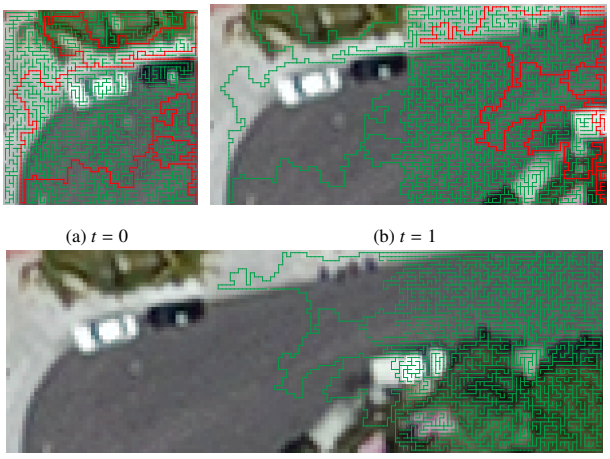
Both cases lead to a contradiction. In fact suppose that e' was originally in $\mathcal{MST}(\mathcal{G}_{t-1})$, then $\mathcal{MST}(\mathcal{G}_{t-1}) \cup \{e\}$ contains a cycle that pass through e and e'. If $w_e < w_{e'}$ $\mathcal{MST}(\mathcal{G}_{t-1})$ would have chosen e instead of e' but it is not the case. On the other hand, if e' is contained in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t) \cap B_t$, then we can find e'' that belongs to $E_{\mathcal{G}_{t-1}}$ but not in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, such that $\mathcal{T}_t \cup \{e''\}$ has a cycle containing e' and e'' (see red dashed line in figure 3). Since e'' is not contained in $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$ we can deduce that $w_{e''} \geq w_{e'}$, and thus $w_{e''} > w_e$. As in the previous case this lead to a contradiction because it implies that $\mathcal{MST}(\mathcal{G}_{t-1}) \cup \{e\}$, with $e \in \mathcal{G}_{t-1}$ contains a MST $\mathcal{T}'$ such that $cost(\mathcal{T}') < cost(\mathcal{MST}(\mathcal{G}_{t-1}))$.

□

We conclude this section showing an interesting insight of the proposed methods that will be useful for possible applications. As the reader may have already noticed, at the end of each iteration t, we decompose the $\mathcal{MST}(\mathcal{G}_t)$ in two disjoint parts: a) $E_{\mathcal{G}_t}$, a graph made of all edges in $\mathcal{MST}(\mathcal{G}_t)$ that may form cycles when the new block arrives. b) $\mathbf{F}_t = \mathbf{T}_t - E_{\mathcal{G}_t}$, a forest made by all the rest of the graph.

The first graph, is indeed made by edges that we call *unstable*. Mostly because we could eliminate some of them in the next step $t + 1$. The second graph is made by edges that we call *stable*, since they will belong to all MSTs from now on. This is important for two reasons. 1) At each step the memory footprint is reduced by discarding edges that are no longer

necessary to compute further MSTs. 2) The *stable* edges can be used for further tasks as we will see below. In Figure 4, we report an example that shows the evolution of the *stable* + *unstable* decomposition of minimum spanning trees through the time. In green we represent the forest $\mathbf{F}_t$ over time, while in red the graph $E_{\mathcal{G}_t}$.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$

Fig. 4: An example of *stable* + *unstable* decomposition of minimum spanning tree. The green graph is the forest $\mathbf{F}_t$ that contains only *stable edges*, while the red graph is $E_{\mathcal{G}_t}$ that contains only *unstable edges*. (b-c) Pixels without edges are stable, so is possible to store that part of the graph and do not need to consider in following intervals.

4. Benchmarks

We conducted our experiments on a high resolution image taken from the site `http://www.gigapan.com/`. The image is (12000×47196) pixels high resolution photo of Planet Mars' surface[2]. The CPU of the machine used for the tests is Intel 3.00 GHz Xeon with 32GB RAM. We considered the case as shown in Section 2 where blocks of an image stream horizontally.

As first benchmark, we measured how long our methods took to build the MST for the complete image and we compared this against the *brute-force* method, that is the method that load in memory all the image and then compute the MST using Kruskal approach. We make the image stream in blocks B_t of size 12000×4000 pixels, and at each interval t we measure the time the algorithms take to compute $\mathcal{MST}(\mathcal{I}_t)$. In Figure 5 we report the results. During the experiment we remarked that the *brute-force* method is faster compared to our algorithms until the image size reaches the limit to fit entirely in RAM. Hence, the reader should be aware that on a Turing Machine with unlimited resources the *brute force* algorithm would be faster. In our machine, the image size limit is about 240 megapixels (Mpx). After that, the *brute-force* method needs to use *swap memory*, and its runtime grows until the image size reach about 380 Mpx, when the image size makes the program crash. On the contrary, thanks to the *stable* + *unstable* decomposition that reduces the memory footprint of Algorithms 2 and 3, the execution times of our methods grow quasi-linearly with the image size, and they are able to treat images of bigger sizes.

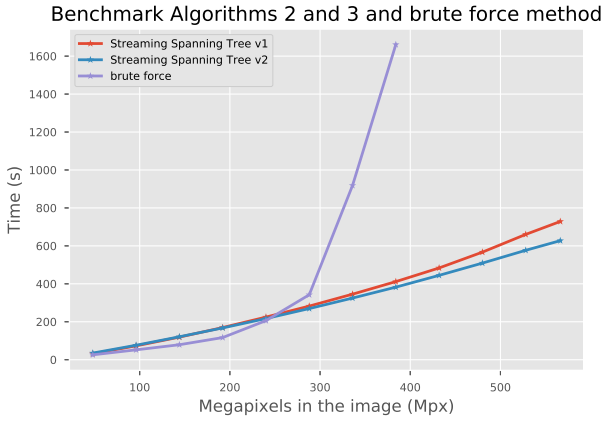


Fig. 5: Runtime of Algorithm 2, Algorithm 3 and *brute-force* algorithm on a (12000×47196) pixels image of Planet Mars' surface.

Furthermore, in order to better understand how the size of the blocks B_t affects the runtime, we measured the execution times of our methods with different sizes for streaming blocks that are 12000×4000, 12000×8000 and 12000×12000 pixels. We report the results obtained in Figure 6. As the reader can see, Algorithm 3 performs better than Algorithm 2 in the general case. Moreover, we noted an improvement in the execution times to treat the entire image when we use block sizes of 12000×8000 pixels compared with block size of 12000×4000 pixels. This is because, using bigger blocks, we reach the entire image dimensions in fewer iterations and we spend less time updating the MSTs. Nevertheless, using blocks of bigger sizes also increases the quantity of memory needed at each iteration and thus increases the risk to use *swap memory* that slows down the overall execution time. For this reason, in the case of Algorithm 3 we do not remarked the same improvement when we use blocks size of 12000×12000 pixels compared with the ones of 12000×8000 pixels. Thus, the ideal block size should be a trade-off between the footprint of the block B_t and the number of iterations needed to treat the entire image.

We end this section by analysing the time complexity of Algorithm 3. Clearly this relies on the algorithm used to compute $\mathcal{MST}(E_{\mathcal{G}_{t-1}} \cup B_t)$, and it depends on the cardinality of sets $E_{\mathcal{G}_{t-1}} \cup B_t$ for all $0 \leq t \leq T$, that are bounded by $|B_t| \leq |E_{\mathcal{G}_{t-1}} \cup B_t| \leq |B_t| + |\mathcal{MST}(\mathcal{G}_{t-1})|$. The best case is indeed when $E_{\mathcal{G}_{t-1}}$ corresponds to the frontier, and the worst case is when $E_{\mathcal{G}_{t-1}} \equiv \mathcal{MST}(\mathcal{G}_{t-1})$, as for example, a graph $\mathcal{MST}(\mathcal{G}_{t-1})$ shaped like a comb facing the frontier. We observe that in this second case the memory complexity of our methods is the same as *brute force* method. Hence, assuming for example to use Kruskal algorithm to compute the MST, the time complexity c of computing a MST for a graph $\mathcal{G}$ is: $O\Big(\frac{(T+1)}{2}(2m\log(m)+N)\Big) \leq c \leq O\Big(m\log(m)+\log(H(m+nT))+\frac{N(T+1)}{2}\Big)$, where N is the number of nodes in $\mathcal{G}$, n, m the number of nodes and edges respectively in each block, T is the number of steps, and $H(m+nT) = \sum_{t=1}^{T}(m+nt)\log(m+nt)$ is the hyperfactorial function.

[2] `http://www.cmm.mines-paristech.fr/~gigli/mars.jpg`

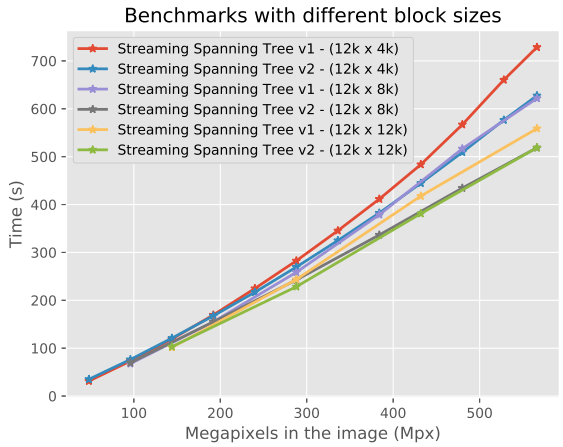


Fig. 6: Runtime of Algorithm 2 and 3 with different block sizes. We used blocks of 12000×4000, 12000×8000 and 12000×12000 pixels.

Doing a similar analysis on Algorithm 2 we find that its complexity, in memory and time, depends on the cardinality of union of unstable edges in both sides of the frontier, i.e, $|E_{\mathcal{G}_{t-1}} \cup E_{B_t}|$. We conclude that the size of the unstable part has a crucial impact on the performance of algorithms.

5. Applications

In this section we introduce three applications of our algorithm to image segmentation. In particular, we exploited the *stable + unstable* decomposition of our methods to implement a streaming version of λ-quasi-flat zones (Najman et al., 2013)(Zanoguera and Meyer, 2002), watershed-cuts (Cousty et al., 2009) and constrained connectivity (Soille, 2008). However, other approaches as hierarchical watershed Cousty and Najman (2011), stochastic watershed Angulo and Jeulin (2007) could be also considered in our framework. In the applications, we use slightly different versions of Algorithms 2 and 3, that each time t return the stable forest $\mathbf{F}_t$ and the unstable graph $E_{\mathcal{G}_t}$. Remember that the MST for the graph $\mathcal{G}_t$ can be obtained as $\mathcal{MST}(\mathcal{G}_t) = \cup_{s=0}^{t} \mathbf{F}_s \cup E_{\mathcal{G}_t}$. In order to validate the streaming versions of the previously mentioned methods, we applied them to the case of horizontal streaming as described in Section 2. We report in Figure 7 the remote sensing image used for our experiments, that we split in three blocks. The blue dashed lines represent the frontiers between two consecutive blocks, while red pixels are the markers used for the watershed-cuts.

Fig. 7: Image used to validate streaming version of the segmentation methods. The image has been split in three blocks (see blue dashed lines) and the blocks stream from left to right. As explained in Section 2 two consecutive blocks share a column of pixels. In red, the pixels used as markers for watershed-cut.

5.1. λ-quasi-flat zones

Let recall the definition of λ-quasi-flat zone hierarchy. Suppose I be an image and consider $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ its associated weighted graph (as seen in Section 2). Given an integer $\lambda \in \mathbb{N}$ we can extract from the graph $\mathcal{G}$ a subgraph $\mathcal{G}_\lambda = (\mathcal{V}, \mathcal{E}_\lambda, \mathcal{W})$ such that $\mathcal{E}_\lambda = \{e \in \mathcal{E} | \mathcal{W}(e) < \lambda\}$, that is the graph $\mathcal{G}_\lambda$ is obtained from $\mathcal{G}$ removing all the edges whose weight is equal or greater than λ. The set $P_\lambda = \{C_0, \ldots, C_{n_\lambda}\}$ made of all connected components of $\mathcal{G}_\lambda$ is a partition of the set of nodes $\mathcal{V}$. The connected components are also called *lambda-quasi-flat zones*, since the variation between two neighboring nodes in a connected component does not exceed λ. Remark that if $\lambda_1 \leq \lambda_2$, then the partition obtained using λ_1 is finer than the one obtained using λ_2. That is, for each connected component $C \in P_{\lambda_1}$ it exists a connected component $C' \in P_{\lambda_2}$ such that $C \subseteq C'$. In that case, we write $P_{\lambda_1} \preceq P_{\lambda_2}$ to indicate that partition P_{λ_1} is finer than partition P_{λ_2}. By making varying the values of lambda from zero to $\max(\mathcal{W})$ we obtain a sequence of partitions such that $P_0 = \mathcal{V} \preceq \ldots \preceq P_{|\mathcal{E}|-1} = \{\mathcal{V}\}$, that is the *$\lambda$-quasi-flat zone hierarchy*. Theorem 4 in Cousty et al. (2018) states that we obtain the same result thresholding the edges of the minimum spanning tree of $\mathcal{G}$. For this reason the algorithms presented above could be suitable to compute a level of λ-quasi-flat zone hierarchy in a streaming fashion, exploiting the *stable + unstable* decomposition. Let explain it using the example in Figure 7. At time $t = 0$ we get $\mathcal{MST}(\mathcal{G}_0) = \mathbf{F}_0 \cup E_{\mathcal{G}_0}$. Now, removing the edges in $\mathcal{MST}(\mathcal{G}_0)$ larger than λ, we obtain $C_0^{(0)}, \ldots, C_{n_0}^{(0)}$ connected components. We call *unstable connected component* a region C containing nodes on the *frontier*. On the contrary, we classify as *stable connected component* a region C that does not contain any node in the *frontier*.

In fact, it is straightforward that a connected component containing nodes on the frontier can change in the following iteration, for example it can expand and include nodes of future blocks. We conclude that, at each time t, we can assign a label only to *stable connected components*. Conversely, we need to keep in memory the *stable edges* contained in *unstable connected components*. We call residual graph $\mathbf{R}_0$, the graph made by nodes contained in *unstable connected components* and *stable edges* contained in them at time $t = 0$. Thank to this decomposition, to get the connected components in future intervals of time, we do not need the entire $\mathcal{MST}(\mathcal{G}_t)$, but is sufficient to consider the graph $\mathbf{R}_{t-1} \cup \mathbf{F}_t \cup E_{B_t}$ and threshold edges larger than λ from it. To summarize, at each time t we do:

1. Consider the graph $\mathbf{G}_t = \mathbf{R}_{t-1} \cup \mathbf{F}_t \cup E_{B_t}$ where $\mathbf{R}_{t-1}$ is the residual graph obtained at previous step.
2. Threshold all edges in $\mathbf{G}_t$ larger than a given λ, obtaining $C_0^{(t)}, \ldots, C_{n_t}^{(t)}$ connected components.
3. Assign a label only to components $C_j^{(t)}$ whose pixels are not in the frontier. In Figure 8 these regions are represented by non-black colors. Mark as *unstable connected components* the regions $C_j^{(t)}$ containing nodes in the frontier. In Figure 8 (a,b) these regions are the black pixels.
4. Assign to $\mathbf{R}_t$ the *stable edges* contained in *unstable connected components* at iteration t.

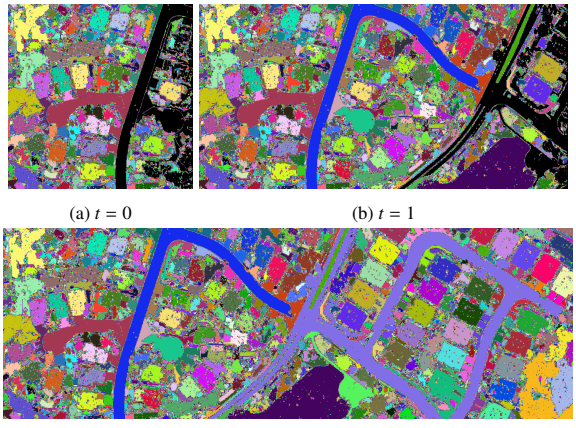
Figure 8 reports the result of the procedure above applied on Figure 7.

5.2. Watershed cuts

Another popular method to segment an image that exploit the minimum spanning tree is the watershed cut (Cousty et al., 2009). This method needs a set $M = \{p_0, \dots, p_k\}$ of pixels called *markers*. At the end of the process each of the marker will be contained in a different region of the segmented image. Basically, the method takes as input a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ and a set M of markers and proceeds in the following way:

1. add to the set of nodes $\mathcal{V}$ as special node z called *well*, that is $\mathcal{V}' = \mathcal{V} \cup \{z\}$
2. for each marker p, add an edge (p, z) to the set of edges $\mathcal{E}$, whose weight is $m - 1$ (with $m = \min_{e \in \mathcal{E}} \mathcal{W}(e)$), i.e. $\mathcal{E}' = \mathcal{E} \cup \{(p, z) | p \in M\}$
3. compute a minimum spanning tree $\mathcal{T}'$ of the graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \mathcal{W}')$
4. return the connected components $C_0, \dots, C_k$, of the subgraph $\mathcal{F} \subseteq \mathcal{T}'$ restricted only to nodes in $\mathcal{V}$.

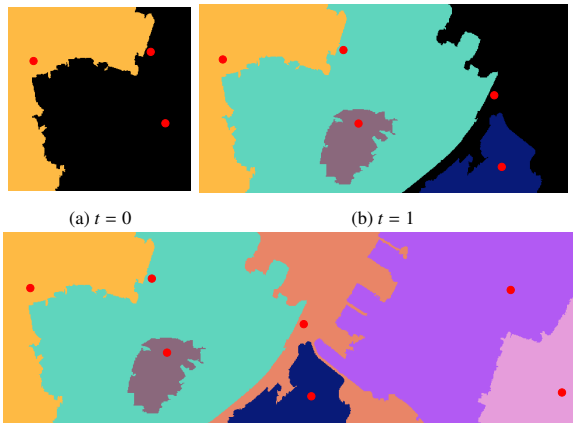
Remark that the connected components are the regions of our segmentation and that a marker is contained in each of them. In fact, each marker p is connected in $\mathcal{G}'$ to the well z with an edge whose weight is minimum. Necessarily those edges will be in the minimum spanning tree $\mathcal{T}'$ of the extended graph $\mathcal{G}'$, and for this reason each path in the MST that connects two markers must pass by the well node z. We conclude that in the subgraph $\mathcal{F} \subseteq \mathcal{T}'$ restricted to nodes $\mathcal{V}$, two markers must belong to different connected components.
It is possible to derive a streaming version of the watershed cuts like we did to get *λ-quasi-flat zones*. Once again we use the example in Figure 7 to illustrate it. At time $t = 0$, we obtain the segmentation as connected components of subgraph $\mathcal{F}' \subseteq \mathcal{MST}(\mathcal{G}'_0)$ as just seen. Also in this case, we split these regions between *unstable* and *stable* based on the criteria of whether they contains frontier nodes or not. Successively, we assign a label only to *stable connected components*, and we retrieve the *residual graph* R_0, that is the graph composed by the nodes in the *unstable connected components* with *stable edges*.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$

Fig. 8: An example of one level of λ-quasi-flat zones in streaming, with $\lambda = 10$ for image in Fig. 7. Black pixels in Figures (a) and (b) are those that do not have a stable label in that iteration.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$

Fig. 9: Watershed cuts in streaming for image in Fig. 7. Red pixels in the images are the markers of the segmentation. Black pixels in Figures (a) and (b) are the connected components that do not have a stable label in that iteration.

Similarly to the previous subsection, to get the connected components in future intervals of time, we do not need the entire $\mathcal{MST}(\mathcal{G}'_t)$, but is sufficient to consider the graph $\mathbf{R}_{t-1} \cup \mathbf{F}'_t \cup E'_{B'_t}$. To summarize, let $\mathcal{G}_t$ a streaming of graph. At each time t we do:

1. Consider the graph $\mathbf{G}'_t = \mathbf{R}_{t-1} \cup \mathbf{F}'_t \cup E_{B'_t}$, where $\mathbf{R}_{t-1}$ is the residual graph obtained at previous step.
2. Compute $C_0^{(t)}, \dots, C_{n_t}^{(t)}$ connected components of the subgraph obtained removing well z and its connections from $\mathbf{G}'_t$.
3. Assing a label exclusively to components $C_j^{(t)}$ whose pixels are not in the frontier. In Figure 9 these components are non-black regions of the image. Mark as *unstable connected components* the regions $C_j^{(t)}$ that contain at least one node in the frontier. In Figure 9 (a,b): these regions are black pixels.
4. Assign $\mathbf{R}_t$ as the graph made by nodes in *unstable connected components* at iteration t with *stable edges*.

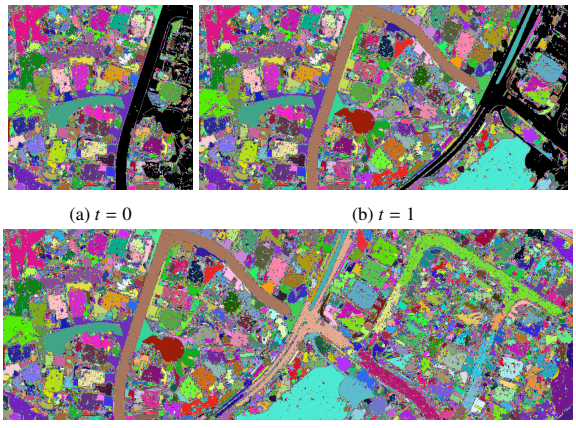
5.3. (α, ω)-constrained connectivity

Lastly, we conclude this section introducing a third application of our streaming methods. This time we show a streaming version of the (α, ω)-constrained connectivity method. Introduced by Soille (2008) this method extends the concept of α-quasi-flat zones and tackles the problem of chaining-effect Soille (2011). In fact, it can happen that distinct objects in the image are separated by one or more transitions going in steps having an intensity height less than or equal to λ. It follows that those objects fall in the same λ-quasi-flat zone even though they are distinct. Essentially, the idea proposed is to introduce a connectivity index to measure the degree of connection of a connected component. Briefly, let I be a grayscale image, and consider a λ-quasi-flat zone C. We define the *range* of the quasi-flat zone $R(C)$ as the biggest difference of intensity among two pixels in C, *i.e.*, $R(C) = \max_{p,q \in C} |I(p) - I(q)|$. In the original paper (Soille, 2008) proposed to use the range of a connected component as a measure of connection, but it could

be any predicate with a non-decreasing property on *λ-quasi-flat zones* such as area or volume of *λ-quasi-flat zones*. However, hereunder we recall its original definition. Given a pixel p, the (α, ω)-connected component of p is the largest λ-quasi-flat zone containing p such that $\lambda \leq \alpha$ and with a range less than ω,

$$(\alpha, \omega) - CC(p) = \max_{\lambda} \Big\{ \lambda - CC(p) | \lambda \leq \alpha \text{ and } R(\lambda - CC(p)) \leq \omega \Big\},$$

where $\lambda - CC(p)$ is the λ-quasi flat zone that contains p. Moreover, two pixels p and q are (α, ω)-*connected* if and only if $q \in (\alpha, \omega) - CC(p)$. It turns out that the relation *"is (α, ω)-connected"* is an equivalence relation and thus it generates a unique partition of the image definition domain. Let now discuss how a streaming version of MST can be used to obtain the (α, ω)-constrained connectivity for a stream of images. We implemented it in a straightforward manner. Basically, at each interval t we first compute the stable α-quasi-flat zones of the image $\mathcal{I}_t$ as in the previous subsection, and then for each stable α connected components we extract the (α, ω) connected components contained in it. In Figure 10 we report the result obtained on the test image in 7.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$

Fig. 10: An example of (α, ω)-constrained connectivity in streaming, with $\alpha = 10$ and $\omega = 150$ for image in Fig. 7. Black pixels in Figures (a) and (b) are those that do not have a stable label in that iteration.

6. Conclusions

This paper introduced two methods for the computation of a minimum spanning tree for graph streaming. We have shown empirically that their execution time grows quasi-linearly with image size. Finally, we have shown how to apply these methods to segmentation tasks. In particular how they can be used to extract a level of λ-quasi-flat-zones hierarchy for image streaming. The main advantage of our proposed algorithm is that at each time the MST is decomposed in two parts. The *stable* part can be stored or used in further tasks. As shown in the case of segmentation. The second one, the *unstable*, is kept in memory. Since it may contain edges that could be removed from MST as future information arrives. Thanks to this decomposition we can reduce the memory necessary to compute the MST and treat images of bigger sizes.

References

Angulo, J., Jeulin, D., 2007. Stochastic watershed segmentation., in: ISMM 2007, pp. 265–276.

Bao, L., Song, Y., Yang, Q., Yuan, H., Wang, G., 2014. Tree filtering: Efficient structure-preserving smoothing with a minimum spanning tree. IEEE Trans. Im. Proc. 23, 555–569.

Couprie, C., Grady, L., Najman, L., Talbot, H., 2011. Power watershed: A unifying graph-based optimization framework. IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 1384–1399.

Cousty, J., Bertrand, G., Najman, L., Couprie, M., 2009. Watershed cuts: Minimum spanning forests and the drop of water principle. IEEE Transactions on Pattern Analysis and Machine Intelligence 31, 1362–1374.

Cousty, J., Najman, L., 2011. Incremental algorithm for hierarchical minimum spanning forests and saliency of watershed cuts, in: ISMM, Springer. pp. 272–283.

Cousty, J., Najman, L., Kenmochi, Y., Guimarães, S., 2018. Hierarchical segmentations with graphs: quasi-flat zones, minimum spanning trees, and saliency maps. Journal of Mathematical Imaging and Vision 60, 479–502.

Fehri, A., Velasco-Forero, S., Meyer, F., 2017. Prior-based hierarchical segmentation highlighting structures of interest, in: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, Springer. pp. 146–158.

Gazagnes, S., Wilkinson, M.H., 2019. Distributed component forests in 2-d: Hierarchical image representations suitable for tera-scale images. International Journal of Pattern Recognition and Artificial Intelligence , 1–22.

Gigli, L., Velasco-Forero, S., Marcotegui, B., 2018. On minimum spanning tree streaming for image analysis, in: 2018 25th ICIP, IEEE. pp. 3229–3233.

Kim, W., 2009. Parallel clustering algorithms: survey. Parallel Algorithms, Spring 34, 43.

Li, S., Dragicevic, S., Castro, F.A., Sester, M., Winter, S., Coltekin, A., Pettit, C., Jiang, B., Haworth, J., Stein, A., et al., 2016. Geospatial big data handling theory and methods: A review and research challenges. ISPRS Journal of Photogrammetry and Remote Sensing 115, 119–133.

Ma, Y., Wu, H., Wang, L., Huang, B., Ranjan, R., Zomaya, A., Jie, W., 2015. Remote sensing big data computing: Challenges and opportunities. Future Generation Computer Systems 51, 47–60.

Matas, P., Dokladalova, E., Akil, M., Grandpierre, T., Najman, L., Poupa, M., Georgiev, V., 2008. Parallel algorithm for concurrent computation of connected component tree, in: International Conference on Advanced Concepts for Intelligent Vision Systems, Springer. pp. 230–241.

Meyer, F., 2001a. Hierarchies of partitions and morphological segmentation, in: International Conference on Scale-Space Theories in Computer Vision, Springer. pp. 161–182.

Meyer, F., 2001b. An overview of morphological segmentation. International journal of pattern recognition and artificial intelligence 15, 1089–1118.

Meyer, F., 2015. The waterfall hierarchy on weighted graphs, in: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, Springer. pp. 325–336.

Najman, L., Cousty, J., Perret, B., 2013. Playing with kruskal: Algorithms for morphological trees in edge-weighted graphs, in: ISMM, pp. 135–146.

Olman, V., Mao, F., Wu, H., Xu, Y., 2009. Parallel clustering algorithm for large data sets with applications in bioinformatics. IEEE/ACM Transactions on Computational Biology and Bioinformatics 6, 344–352.

Soille, P., 2008. Constrained connectivity for hierarchical image partitioning and simplification. IEEE Transactions on Pattern Analysis and Machine Intelligence 30, 1132–1145. doi:10.1109/TPAMI.2007.70817.

Soille, P., 2011. Preventing chaining through transitions while favouring it within homogeneous regions, in: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, Springer. pp. 96–107.

Tu, W.C., He, S., Yang, Q., Chien, S.Y., 2016. Real-time salient object detection with a minimum spanning tree, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2334–2342.

Wei, X., Yang, Q., Gong, Y., Ahuja, N., Yang, M.H., 2018. Superpixel hierarchy. IEEE Transactions on Image Processing 27, 4838–4849.

Wu, B.Y., Chao, K.M., 2004. Spanning trees and optimization problems. CRC Press.

Zanoguera, F., Marcotegui, B., Meyer, F., 1999. A toolbox for interactive segmentation based on nested partitions, in: Proceedings 1999 ICIP, IEEE. pp. 21–25.

Zanoguera, F., Meyer, F., 2002. On the implementation of non-separable vector levelings. Proc. of VIth ISMM, Sydney, CSIRO , 369–377.