Manipulation of hierarchical segmentation of images based on saliency maps

Gabriel Miranda de Araújo

Supervisor: Prof. Dr. Paulo A. V. de Miranda

August 2020

Contents

Abstract

The segmentation of an image consists of severing a digital image in multiple segments, such as those with similar pixel characteristics, therefore making it possible to analyse each one separately. Alongside this concept, we have the hierarchy of an image, which consists of a sequence of its partitions, with consecutive partitions having the refinement property. Hierarchies can be represented either by a dendrogram, graph or a minimum spanning tree. Throughout this work, we will explore the graph representation of hierarchies; more specifically, we'll implement both saliency maps and Khalimskygrids as designated data structures to aid in the manipulation and visualization of image hierarchies, respectively.

In 2017, Guimarães et al. proposed an efficient digital image hierarchical segmentation algorithm and demonstrated the correspondence between hierarchies and saliency maps, in order to identify and construct the partitions that make up a hierarchy. Since these concepts are being applied in a graph representation, it opens up the possibility to combine them with the Max-tree data structure, enabling the efficient calculation of extinction values of image regions, and using these values to rearrange its hierarchy, possibly leading to performance gains in automatic segmentation algorithms.

In this work, following the above strategy, we evaluate the rearrangement of the hierarchy produced by a recent method of unsupervised segmentation by Oriented Image Foresting Transform (UOIFT). UOIFT, as proposed by Bejar et. al in 2020, generates a hierarchical image partition by successive optimum cuts in graphs, that can be tailored to different objects, according to their boundary polarity. Here, we reorganize its hierarchy in order to incorporate other high-level information of the objects of interest, in addition to the boundary polarity, such as their sizes. The results are demonstrated in natural and medical images.

Resumo

A segmentação de uma imagem consiste em fatiar uma imagem digital em diversos segmentos, com estes tendo características em comum entre eles, possibilitando, portanto, uma análise singular de cada um. Paralelo a esse conceito, temos também a hierarquia de uma imagem, que é composta por um conjunto de partições, determinadas por critérios arbitrários, e cada uma podendo ser representada por um dendograma, grafo ou *minimum spanning tree*. Ao longo desse trabalho, exploraremos a representação das hierarquias através de grafos; mais especificamente, serão implementadas grades de Khalimsky e mapas de saliência como as estruturas de dados padrão para manipulação de hierarquias de imagens.

Em 2017, Guimarães et al. propuseram um método eficiente de segmentação hierárquica de imagens digitais, simultâneo à uma correspondência de hierarquias e mapas de saliência, como forma de identificar e construir partições que compõem uma hierarquia. Dado que esses conceitos estão sendo aplicados em representações de grafos, é aberta a possibilidade para combiná-los com a estrutura de dados Max-tree, permitindo então o cálculo eficiente de valores de extinção de regiões na imagem, para, posteriormente, utilizá-los como critério para a manipulação da hierarquia da imagem em questão, possivelmente ofertando ganhos de desempenho em algoritmos de segmentação automática.

Neste trabalho, seguindo a estratégia acima, avaliamos o rearranjo da hierarquia produzida por um método recente de segmentação não supervisionada por Oriented Image Foresting Transform (UOIFT). UOIFT, conforme proposto por Bejar et. al em 2020, gera uma partição hierárquica de imagens por sucessivos cortes ótimos em grafos direcionados, que podem ser ajustados a diferentes objetos, de acordo com suas polaridades de borda. Aqui, reorganizamos sua hierarquia para incorporar outras informações de alto nível dos objetos de interesse, além da polaridade de borda, tal como seus tamanhos. Os resultados são demonstrados em imagens naturais e médicas.

1 Introduction

Hierarchical representation methods of images, video and multimedia analysis seek to explore visual representation as a region oriented space. These methods produce a hierarchy of partitions, made by a set of partitions in different levels of detail, in which the representation of higher, more refined levels, is nested, in comparison to their coarser counterparts. This type of Data Structure has been successfully applied in remote sensing, object detection and human action recognition.

Non-hierarchical methods, like the one proposed by Felzenszwalb and Huttenlocher [6], that use a similarity measure to merge two adjacent regions in order to form a new segment, can also be transformed, without loss of quality, into hierarchical methods, through the incorporation of new properties [7].

The present work is related to the CAPES/COFECUB project, entitled *Hierarchical Graph-based Analysis of Image, Video and Multimedia Data*, of which the supervisor Paulo Miranda is a collaborator. The main goal of this broader project is to advance in the state-of-the-art on *hierarchy of partitions* taking into account aspects of efficiency, quality, making hierarchical and interactivity, as well as the use of hierarchical information to help in the information extraction and the label propagation. Moreover, it intends to investigate hierarchical visualization of all, image, video and multimedia, by using countour saliency maps. Finally, it intends to explore the criteria for hierarchical comparison and for hierarchical combination taking into account their contour saliency maps and learning methods. The results of these studies can be used for solving several applications like human action recognition, pornography detection, image and video region labeling, multimedia label propagation, image and video inpainting, among others.

In this work, we address the particular problem of how to manipulate and reorganize a hierarchical image segmentation based on saliency maps.

1.1 Motivation

Despite the several ways for computing hierarchies of partitions, developing efficient and effective methods is not an easy task due to the semantic information which is needed for a segmentation. In [2], a new method UOIFT (*Unsupervised Oriented Image Foresting Transform*), based on hierarchical segmentation of directed graphs, which incorporates the boundary polarity data of target objects, has been proposed, in order to promote regions that resemble such objects to upper levels of the hierarchy, thus facilitating their isolated analysis. Nevertheless, more studies seeking for a better understanding of other relevant high-level attributes are still needed, in an effort to reduce the overall quantity of false positives.

In this work, we evaluate the rearrangement of the hierarchy produced by UOIFT in order to incorporate other high-level information of the objects of interest, in addition to the boundary polarity, such as their sizes. The results are demonstrated in natural and medical images.

1.2 Objectives

The visual representation of a hierarchy of an image can be constructed via a Khalismsky grid of its saliency map [5]. In this work, we seek to explore the reorganization of image partition hierarchies, by means of modifying its respective saliency map graphs, resulting in entirely new hierarchies, via the quasi-flat zone mapping of the modified maps. The goal is to highlight objects of interest, promoting them to higher levels in the hierarchy, without requiring too much processing power. This work will be developed in majority using the C language.

Although the graph used by UOIFT is a directed graph, given that the graphs of saliency maps are undirected graphs, a viable solution would be to explore the extinction values of their Min-trees [10] in order to rearrange them to obtain new hierarchies. We adopted this solution in this work. Other options are discussed in [4], [3] and [8].

2 Concepts and definitions

In this section, we outline the major concepts and notions required for a complete understanding of this work. The relevant references for further reading will also be provided.

2.1 Graph

A Graph is an ordered pair $G = (V, E)$, with V being a set of vertices, and E the set of edges, disjoint of V. Each element of E is an unordered pair of distinct elements of V; in other words, a set with two vertices. The expression $V(G)$ can be used to make reference to the vertices of a graph G, as $E(G)$ can be used to represent its edges. Given the vertices a, b and edge e, with $\{a, b\} \subseteq V(G)$, $e = \{a, b\}$ and $e \subseteq E(G)$, it can be said that a and b are *adjacent* vertices.

Let G be a graph, and w a map from the set of edges to the set of $\mathbb{R}^+$ of non-negative real numbers; the pair (G, w) is denominated an *edge-weighted graph*. If (G, w) is an *edge-weighted graph*, for any edge e of G, the *weight of* e is written as $w(e)$.

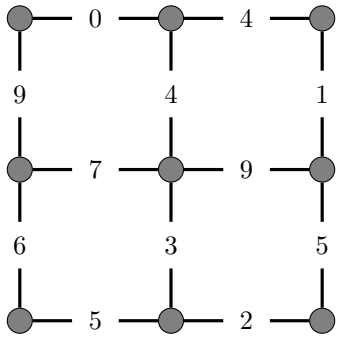


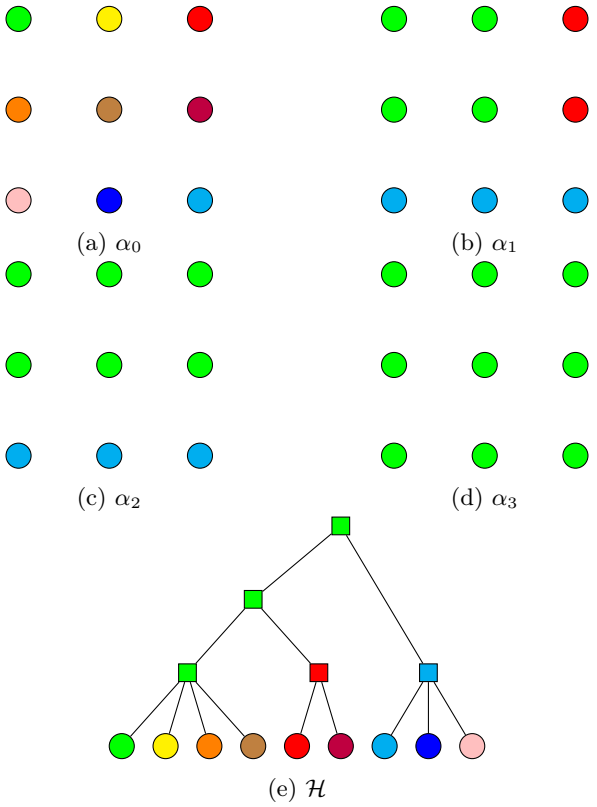
Figure 1: An edge-weighted graph (G, w) with weighted edges.

2.2 Hierarchies

To establish a concrete definition of a hierarchy, we first must build up on the concept of *partitions*. A *partition* of a finite set A is a set α of nonempty disjoint subsets of A whose union is A (that is, $\forall X, Y \in \alpha$, $X \cap Y = \emptyset$ if $X \neq Y$ and $\bigcup_{X \in \alpha} = A$). The elements that make up a partition α are called *regions of* α. If a is part of A, there is a unique region of α that contains a, denoted by $[\alpha]_a$.

Given two partitions α and α' of a set A, it can be stated that α' is a *refinement* of α if any region of α' is contained in a region of α. A *hierarchy* on set A is a sequence $\mathcal{H} = (\alpha_0,...,\alpha_\ell)$ of partitions of A such that $[\alpha]_{i-1}$ is a refinement of $[\alpha]_i$, for any $i \in \{1, ..., \ell\}$. In this described hierarchy $\mathcal{H} =$

$(\alpha_0,...,\alpha_\ell)$, the integer ℓ refers to the *depth of* $\mathcal{H}$. A hierarchy $\mathcal{H} = (\alpha_0,...,\alpha_\ell)$ is considered complete if $\alpha_\ell = \{A\}$ and α_0 contains every single element of A (isto é, $\alpha_0 = \{\{x\} \mid x \in A\}$). The hierarchies studied in this project are complete.



(a) α_0 (b) α_1 (c) α_2 (d) α_3 (e) $\mathcal{H}$

Figure 2: A Hierarchy $\mathcal{H} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$. At every partition α_i, nodes with the same color belong to the same region.

2.3 Quasi-flat zone

Throughout this work, we'll consider a weighted graph (G, w), along with the vertex set V and edge set E of G. We will assume that G is a connected graph, it is also possible to assume, without loss of generality, that the range of w is the set $\mathbb{E}$ of all integers from 0 to $|E| - 1$. The set $\mathbb{E} \cup \{|E|\}$ is, from this point on, referred to as $\mathbb{E}^\bullet$.

Let X be a subgraph of G and let λ be an integer in $\mathbb{E}^\bullet$. The *λ-level set of X (for w)* is the set $w_\lambda(X)$ of all edges of X whose weight is less than λ:

$$w_\lambda(X) = \{e \in E(X) | w(e) < \lambda\}. \tag{1}$$

The *λ-level graph of X (for w)* is the subgraph $w_\lambda^V(X)$ of X whose edge set is

the λ-level set of X and whose vertex set is the one of X:

$$w_\lambda^V(X) = (V(X), w_\lambda(X)). \tag{2}$$

The connected component partition $C(w_\lambda^V(X))$ induced by the λ-level graph of X is called the *λ-level partition of X (for w)*. As demonstrated in [5], for any λ_1 and λ_2 in $\mathbb{E}^\bullet$ such that $\lambda_2 \geq \lambda_1$, the λ_1-level partition of X is a refinement of the λ_2-level partition of X. Hence, the sequence

$$\mathcal{QFZ}(X, w) = (C(w_\lambda^V(X)) | \lambda \in \mathbb{E}^\bullet) \tag{3}$$

of all λ-level partitions of X is a hierarchy. This hierarchy is called the *quasi-flat zone hierarchy* $\mathcal{QFZ}(X, w)$. Figure 3 illustrates the process of constructing $\mathcal{QFZ}(X, w)$.

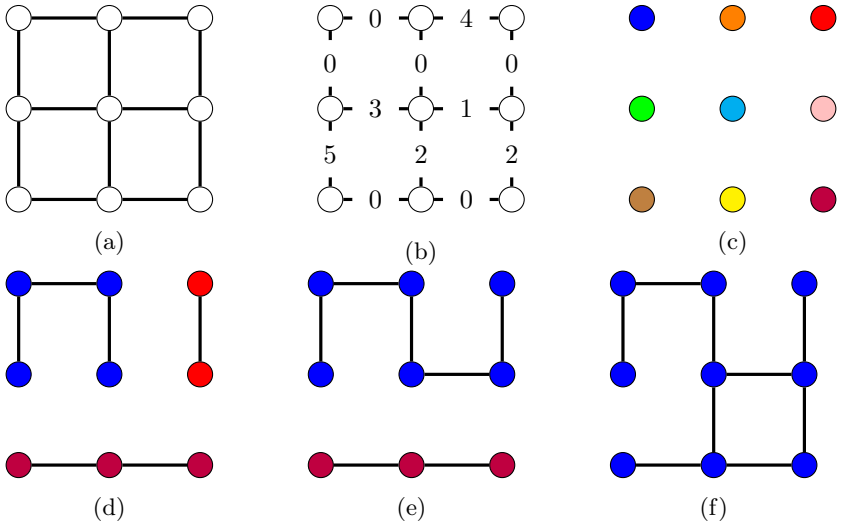


Figure 3: A quasi-flat zone hierarchy. A Graph G in (a), followed by its weight map w in (b). (c, d, e & f) are the λ-level graphs of G, with $\lambda = 0, 1, 2, 3$, respectively.

2.4 Saliency map

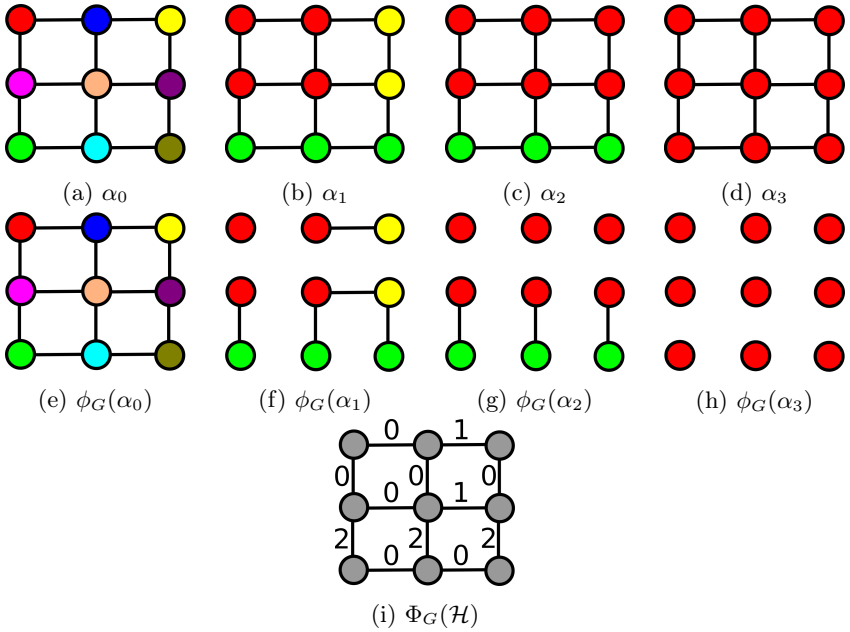
We have seen that any weighted graph on the edges induces a connected hierarchy of partitions (called the quasi-flat zone hierarchy). In this section, we address the inverse problem, that is, given a connected hierarchy $\mathcal{H}$, find a map w so that the quasi-flat zone hierarchy for w is precisely $\mathcal{H}$. We will see that saliency maps provide a solution to this problem [5].

Let α be a partition of V, the cut of α (in G), denoted by $\phi_G(\alpha)$, is the set of edges of G formed by two vertices in different regions of α, that is, $\phi_G(\alpha) = \{\{x, y\} \in E \mid [\alpha]_x \neq [\alpha]_y\}$.

Let $\mathcal{H} = (\alpha_0, \ldots, \alpha_\ell)$ be a hierarchy on V. The *saliency map of* $\mathcal{H}$ is the map $\Phi_G(\mathcal{H})$ from E to $\{0, \ldots, \ell\}$ such that the weight of any edge e for $\Phi_G(\mathcal{H})$ is the maximum value λ for which e belongs to the cut of α_λ:

$$\Phi_G(\mathcal{H})(e) = \max\{\lambda \in \{0, \ldots, \ell\} \mid e \in \phi_G(\alpha_\lambda)\} \tag{4}$$

An example is shown in Figure 4.



(a) α_0 (b) α_1 (c) α_2 (d) α_3

(e) $\phi_G(\alpha_0)$ (f) $\phi_G(\alpha_1)$ (g) $\phi_G(\alpha_2)$ (h) $\phi_G(\alpha_3)$

(i) $\Phi_G(\mathcal{H})$

Figure 4: (a-d) Example of a connected hierarchy $\mathcal{H} = (\alpha_0, \ldots, \alpha_\ell)$ with $\ell = 3$ in a 4-neighborhood graph. (e-h) The set of edges belonging to the cuts $\phi_G(\alpha_i)$ for each partition α_i of the hierarchy for $0 \leq i \leq \ell$. (i) The resulting saliency map $\Phi_G(\mathcal{H})$.

2.5 Khalimsky grid

When the graph is given by the 4-adjacency relation, a saliency map can be visualized through Khalimsky grids, as illustrated in Figure 5.

In 2D, a Khalimsky grid is a set of squares, rectangles, and dots (Figure 5e). Each vertex of the graph is identified to a square with a null value of the Khalimsky grid. Then, each edge $\{x, y\}$ and its weight in the saliency map are identified to the rectangle corresponding to the common side of the two squares identified with x and y. Finally, each dot receives the maximal value of its neighboring rectangles. Since the elements of the Khalimsky grid can be aligned on a square matrix, in this representation, the saliency map can be visualized as an image (Figure 5f).

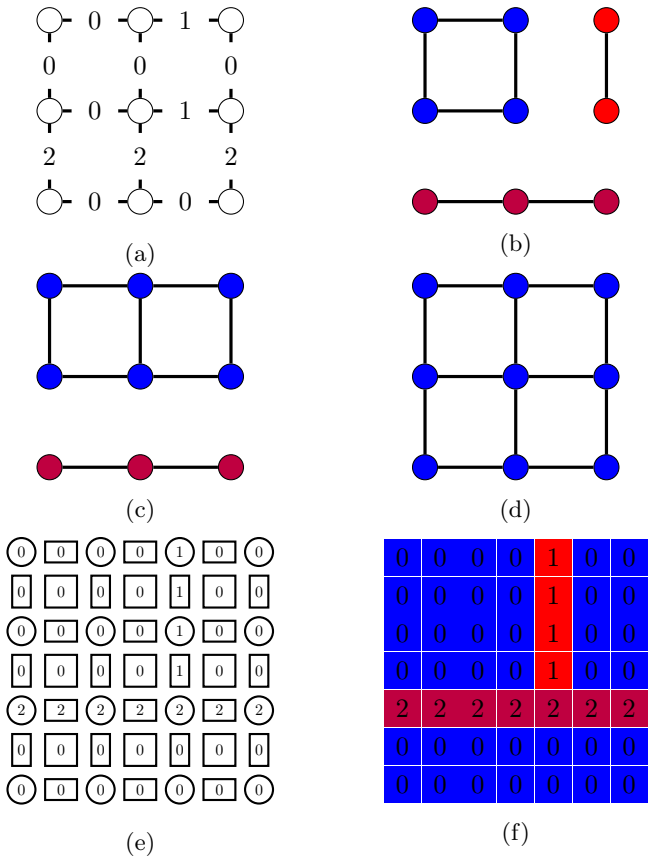


Figure 5: (a) The saliency map $s = \Phi_G(\mathcal{H})$ from Figure 4i of the hierarchy $\mathcal{H}$ from Figures 4a-d. (b-d) The 1-, 2-, and 3-level graphs of G for s, where two vertices belonging to a same connected component are marked with the same color. (e) The corresponding Khalimsky grid of s and (f) its image representation when one considers the 4-adjacency graph.

2.6 Max-tree and extinction values

Let $I : D \to \mathbb{Z}$ denote an image, where D is the image domain. An image I decomposed by threshold forms a set of binary images $B_l : D \to \{0, 1\}$, $l = 0, 1, \ldots, I_{max}$, where $B_l(p) = 1$ if $I(p) \geq l$ and $B_l(p) = 0$ otherwise ($I_{max} = \max_{p \in D} I(p)$).

Given an image I with its respective graph representation, a *component tree* is a representation of the image that describes topological relationships between the connected components of its decomposition by thresholding. Analyzing consecutive levels of this decomposition, we can see that components of the level $l+1$ are contained in components of the level l. A component tree is, therefore, a graph that stores this hierarchy between components. Thinking of the image as a topographic surface, we have that each dome, represented by its respective regional maxima, can be eliminated by pruning its respective branch of the tree. Each dome has different attribute values such as for its height, area and volume that can be used to decide on the selection of the appropriate branches for pruning. Thus, by pruning branches of this tree through appropriate rules, it is possible to generate connected filters (e.g., area opening, volume opening).

Max-tree is a compact representation of the component tree, where each pixel is stored in a single node in the tree, corresponding to the highest level at which it appears in the component tree. This tree is known as the max-tree, since its leaves are always the regional maxima of the image.

A *Min-tree* is the data structure representing the negative image of the Max-tree, obtained by constructing the latter on the complement of an image I. Therefore, since the leaves of a Max-tree were a representation of regional maxima in I, it can be said that the leaves of its related Min-tree store the regional minima of I.

Along with these structures, comes the concept of *extinction values*. For a given attribute (e.g., height, area or volume), the extinction value of a regional maxima of the max-tree corresponds to the value of the attribute filter (max-tree pruning) sufficiently large for the vanishing of the hill. For the 1D image shown on Figure 6, Figure 7 shows one example of max-trees and extinction values.

2.7 Superpixel

Superpixels are compact clusters of connected pixels, locally representing a same image structure, that hold similar characteristics, like pixel intensity, color, texture and position. Since the pixels contained in the same superpixel are considered equal by definition, superpixels primitives have some advantages over simple pixel primitives, like computational efficiency, since the number of primitives are greatly reduced at the superpixel level. This brings great opportunities to alleviate Computer Vision pipelines overhead, by replacing the rigid structure of the pixel grid. Throughout this work, superpixels are used extensively as a first step, mainly due to its performance gains. In this work, the superpixels were computed by the IFT-SLIC method [1].

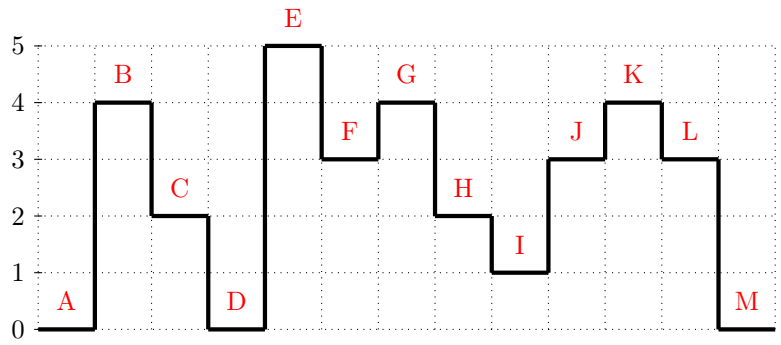
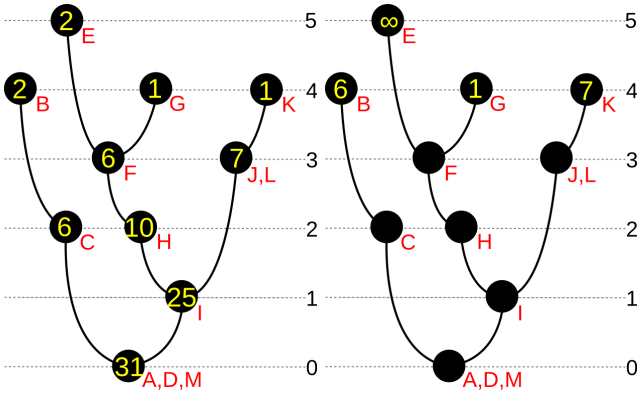


Figure 6: 1D example image



(a) Max-tree with volume (b) Extinction of volume

(c) Max-tree with area (d) Extinction of area

(e) Max-tree with height (f) Extinction of height

Figure 7: Max-trees of the image in Figure 2e. In the first column, attribute values are shown in the nodes for volume, area and height. In the second column, the extinction values are shown in the leaves for different attributes.

3 Algorithms implemented

In this section, we'll discuss both the code and adaptations made around it in order to develop the necessary software for this research.

3.1 Extinction values

The code below allows us to calculate generic extinction values based on the Max-tree data structure:

Algorithm 1: Calculation of extinction values

input : MT_I, E_μ
output: E_μ
Data: An image I Max-tree (MT_I) along with an empty array (E_μ)
Result: Calculated extinction values for all leaves of the tree as an array (E)

foreach $\mathcal{N}^{\mathcal{L}} \in$ *leaves of* MT_I **do**
  $extinction \leftarrow \infty$
  $continue \leftarrow true$
  $\mathcal{N}^{\mathcal{P}} \leftarrow \mathcal{N}^{\mathcal{L}}$
  while *continue* **and** $\exists \mathcal{N}^{\mathcal{L}}$ **do**
    $\mathcal{N}^{\mathcal{A}} \leftarrow \mathcal{N}^{\mathcal{P}}$
    $\mathcal{N}^{\mathcal{P}} \leftarrow$ *parent of* $\mathcal{N}^{\mathcal{A}}$
    if $\exists \mathcal{N}^{\mathcal{P}}$ **and** *(number of children of* $\mathcal{N}^{\mathcal{P}} > 1$*)* **then**
      foreach $\mathcal{N}^{\mathcal{C}} \in$ *(children of* $\mathcal{N}^{\mathcal{P}}$*)* **and** *continue* **do**
        if *(*$\mathcal{N}^{\mathcal{C}}$ *already visited)* **and** $\mathcal{N}^{\mathcal{C}} \neq \mathcal{N}^{\mathcal{A}}$ **and** $\mu(\mathcal{N}^{\mathcal{C}}) \neq \mu(\mathcal{N}^{\mathcal{A}})$*)* **or** $(\mathcal{N}^{\mathcal{C}} \neq \mathcal{N}^{\mathcal{A}}$ **and** $\mu(\mathcal{N}^{\mathcal{C}}) > \mu(\mathcal{N}^{\mathcal{A}})$ **then**
          $continue \leftarrow$ **false**
          $\mathcal{N}^{\mathcal{C}}$ *is marked as visited*
    if $\exists \mathcal{N}^{\mathcal{P}}$ **then**
      $extinction \leftarrow \mu(\mathcal{N}^{\mathcal{A}})$
    ${\mathcal{N}^{\mathcal{L}}}_{ext} \leftarrow extinction$
  $E_\mu(x) \leftarrow \begin{cases} {\mathcal{N}^{\mathcal{L}}}_{ext},\ \forall x \in \mathcal{C}_{\mathcal{N}^{\mathcal{L}}} \text{ and } \mathcal{N}^{\mathcal{L}} \in \text{(leaves of } MT_I\text{)} \\ 0,\ otherwise \end{cases}$
  return E_μ

Starting from the leaves and working its way up to the root of the tree, the *while* loop calculates each leaf's extinction value of the Min/Max-tree. Like previously mentioned, the criteria E_μ used to calculate the extinction values consists in either *height, length, width, area or volume* of a Min/Max-tree of a given image I, as can be seen on [11] in more detail.

3.2 Max-tree construction

Algorithm 2: Tree construction

input : $I, \mathcal{N}_{\mathcal{E}}$
output: MT_i
Data: An image I along with a node value map $\mathcal{N}_{\mathcal{E}}$
Result: Calculated extinction values for all leaves of the tree as a Max-tree data structure of image I (MT_i)

$level[k] \leftarrow$ **false**, $\forall k \in [0, n_{max}]$
$label[k] \leftarrow 0$, $\forall k \in [0, n_{max}]$
$queue[k] \leftarrow \emptyset$, $\forall k \in [0, n_{max}]$
$queue[min(I)].insert(X_m)$ such that $I(x_m) = min(I)$
$status[x] \leftarrow 0$, $\forall x \in E \subset \mathbb{N}^2$
$MT_I \leftarrow \emptyset$
`FLOOD(`$min(I)$`)`
(n)
$x_{\ulcorner} \leftarrow \{\infty, \infty\}$
$x_{\llcorner} \leftarrow \{0, 0\}$
while $queue[n] \neq \emptyset$ **do**
    $p \leftarrow queue[n].$**remove**()
    if $p < x_{\ulcorner}$ **then**
        $x_{\ulcorner} \leftarrow p$
    if $p > x_{\llcorner}$ **then**
        $x_{\llcorner} \leftarrow p$
    foreach $q \in \mathcal{N}_E(p)$ **do**
        if $status[q] = 0$ **then**
            $m \leftarrow I[q]; queue[m].insert(q); level[m] \leftarrow$ **true**
            $status[q] \leftarrow -1$
            while $m > n$ **do**
                $m \leftarrow$ `FLOOD(`m`)`
$m \leftarrow n - 1$
while $m \geq 0$ *and* $(\neg level[m])$ **do**
    $m \leftarrow m - 1$
if $m \geq 0$ **then**
    $MT_I.LINK(\{m, label[m] + 1\}, \{n, label[n] + 1\}, x_{\ulcorner}, x_{\llcorner})$
else
    $MT_I.LINK(\{-1, 1\}, \{\min(I), 1\}, x_{\ulcorner}, x_{\llcorner})$
$level[n] \leftarrow$ **false**
$label[n] \leftarrow label[n] + 1$
return m

As a more traditional approach, this implementation above is based on the works by Lotufo et. al [10] and [11], where, utilizing the watershed transform

method to compute the incremental calculation of the Max-tree attributes, and applying the hierarchical flood shown in [9], we're able to create the desired Tree structure.

3.3 Hierarchy manipulation

Our goal is to reorganize the hierarchy produced by UOIFT [2], in order to incorporate other high-level information on the desired objects.

Although UOIFT is a divisive top-down approach, its output consists of a text file containing the merge history of pairs of nodes on the image graph together with their respective fusion energy values corresponding to the optimal cuts. The UOIFT file contains for each line a pair of pixels and its respective fusion energy. Pixel pairs from the merge history can be seen as edges of a graph, so the merge history forms a tree in the graph. For example, Figure 8a shows the tree resulting from the merge history of the file shown below for an image 4×4, where the nodes are indicated by letters:

```
a  e  8
k  o  4
n  o  2
h  l  2
d  h  1
f  g  1
b  c  0
c  d  0
e  i  0
j  k  0
k  g  0
i  m  0
m  n  0
l  p  0
o  p  0
```

By convention, fusions with zero energy correspond to internal connections of the superpixels (see Figure 8b). Figure 8c shows the corresponding saliency map of the merge tree provided in Figure 8a. Note that the tree in Figure 8a corresponds to the minimum spanning tree (MST) of its saliency map. This tree is created from the merge history file in Lines 1-8 of Algorithm 3.

To reorganize the hierarchy, we first compute its edge-based min-tree, as well as its volume attribute and volume extinction values in Lines 9-13 of Algorithm 3 (see Figures 8d-f). Next, we calculate the highest extinction value on Lines 14-17. For a given node of the graph G, $Tree.map[node]$ returns its corresponding vertex of the min-tree. In Lines 18-22, we obtain the reverse mapping. This code assumes that the min-tree leaves are the first nodes in the min-tree structure. Then a priority queue is created (Line 24) and the leaves of the min-tree are processed in decreasing order of their volume extinction values in the loop of

Line 32. The current leaf has a corresponding pixel p in G (Line 34). At each iteration, the edge with the highest weight is found in G, connecting p in G to some other already processed vertex using a breadth-first search. Its corresponding edge in *cloneGraph* then has its weight replaced by the extinction value of the current iteration (Line 56). In the end, *cloneGraph* contains a tree with modified weights representing a new reorganized hierarchy (Figure 9).

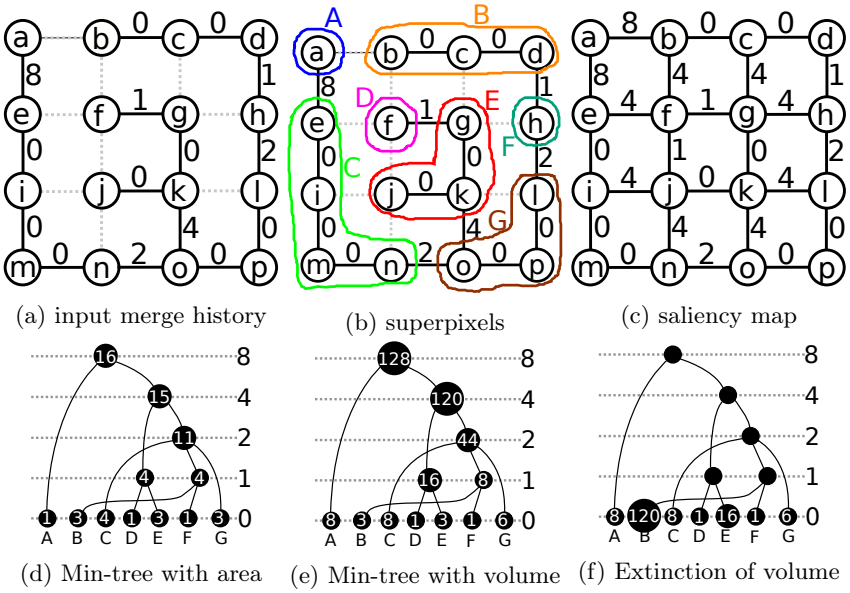
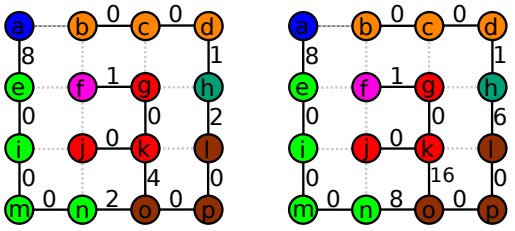


Figure 8: (a) The input merge history of each node on the image graph. (b) By convention, the superpixels are the connected components of the 1-level graph of (a). (c) The saliency map of (a). (d) The min-tree of (a) with area attribute inside the nodes. (e) The min-tree of (a) with volume attribute. (f) The extinction values of the volume attribute inside the leaves.



(a) input hierarchy (b) reorganized hierarchy

Figure 9: (a) The input hierarchy. (b) The volume-based reorganized hierarchy. Vertices belonging to different superpixels are shown in different colors. Note that, in the first hierarchy, the most important edge $\{a, e\}$ only isolates the single blue vertex, while in the new hierarchy the edge $\{k, o\}$ becomes the most relevant, defining the central region composed of four pixels (i.e., pixels f, g, j and k).

Algorithm 3: Hierarchy manipulation - Part 1

input : $nCols$, $nRows$, MH
output: MH'
Data: A text file containing the merge history of each node on the image graph (MH), along with the correct image dimensions ($nCols$, $nRows$)
Result: The reorganized *hierarchy*, utilizing the volume Min-Tree attribute as criteria

```
1  G ← new Graph(nCols × nRows)
2  array[k] ← 0, ∀k ∈ [0, 2]
3  foreach line ∈ MH do
4      k ← 0
5      foreach element ∈ line do
6          array[k] ← element
7          k ← k + 1
8      AddEdge(G, array[0], array[1], array[2])
9  Tree ← EdgeBasedMinTree(G)
10 ComputeHeight(Tree)
11 ComputeArea(Tree)
12 ComputeVolume(Tree)
13 ext ← ComputeExtinctionValue(Tree, volume)
14 extMax ← 0
15 foreach leaf ∈ nleaves(Tree) do
16     if ext[leaf] > extMax then
17         extMax ← ext[leaf]
18 inverseMap[k] ← 0, ∀k ∈nleaves(Tree)
19 foreach node ∈ nnodes(G) do
20     temp ← Tree.map[node]
21     if temp < nleaves(Tree) then
22         inverseMap[temp] ← node
23 cloneGraph ← Clone(G)
24 Q ← createPriorityQ(extMax + 1, leaves(Tree), ext)
```

Algorithm 4: Hierarchy manipulation - Part 2

input : $nCols$, $nRows$, MH
output: MH'
Data: A text file containing the merge history of each node on the image graph (MH), along with the correct image dimensions ($nCols$, $nRows$)
Result: The reorganized *hierarchy*, utilizing the volume Min-Tree attribute as criteria

```
25 foreach leaf ∈ leaves(Tree) do
26     insertElement(Q, leaf)
27 visited ← createBitmap(nnodes(G))
28 FIFO ← createQ(nnodes(G))
29 pred[k] ← NIL ∀k ∈ nnodes(G)
30 Qindex ← getMax(Q)
31 bitMap.Set1(visited, inverseMap[Qindex])
32 while ¬ isEmpty(Q) do
33     Qindex ← getMax(Q)
34     p ← inverseMap[Qindex]
35     push(FIFO, p)
36     pred[p] ← NIL
37     while ¬ isEmpty(FIFO) do
38         p ← pop(FIFO)
39         if get(visited, p) then
40             break
41         foreach node ∈ adjacentNodes(G.nodes[p]) do
42             q ← G.nodes[p].adjacent[node]
43             if pred[p] ≠ q then
44                 pred[q] ← p
45                 push(FIFO, q)
46     reset(FIFO)
47     wmax ← 0
48     q ← p
49     while pred[q] ≠ NIL do
50         w ← getArcWeight(G, pred[q], q)
51         if w > wmax then
52             wmax ← w
53             qmax ← q
54         q ← pred[q]
55     RemoveEdge(G, pred[qmax], qmax)
56     UpdateEdge(cloneGraph, pred[qmax], qmax, ext[Qindex])
57     p ← inverseMap[Qindex]
58     bitMap.Set1(visited, p)
59 return cloneGraph
```

3.4 Khalimsky grid

The next algorithm computes the Khalimsky Grid of a given saliency map, which is received as an input 4-neighborhood graph, as discussed in more detail in [5].

Algorithm 5: Khalimsky grid

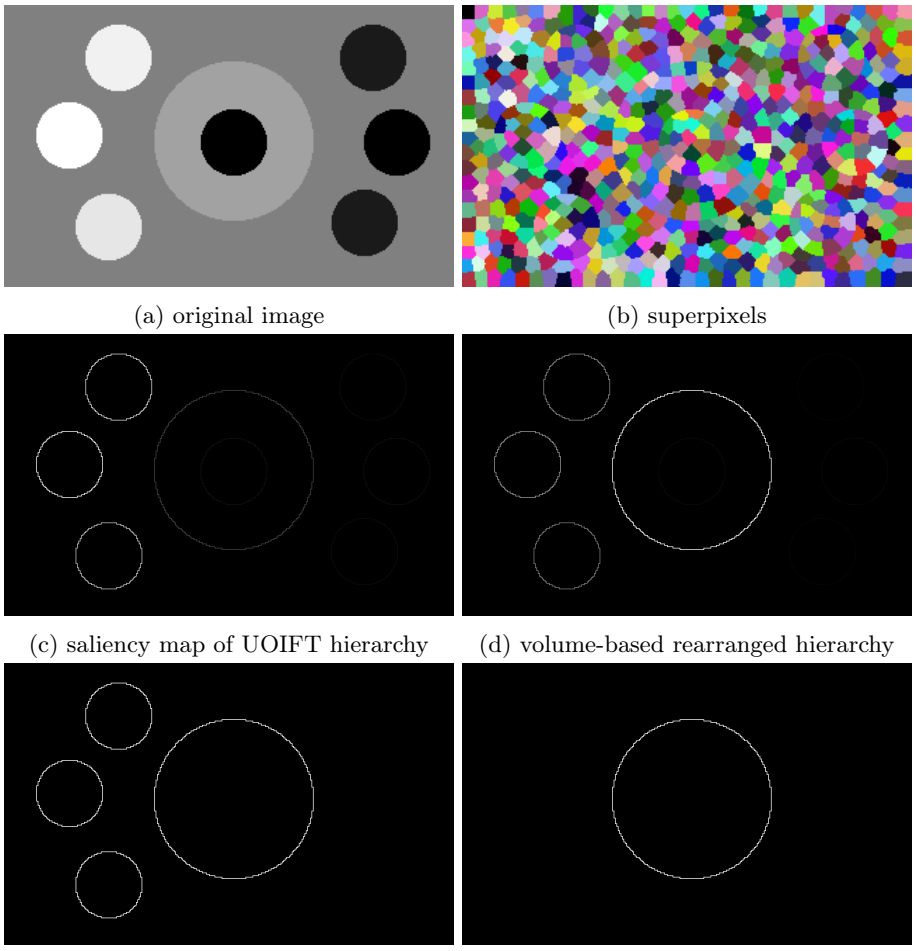
input : *Graph*
output: *grid*
Data: Graph of the image we wish to translate into a Khalimsky grid data structure (*grid*)
Result: Khalimsky grid generated using the original image graph (*Graph*)

```
1  A ← Adjacency(Graph)
2  if nneighbors(A) ≠ 4 then
3      return NULL
4  grid ← Create(ncols(Graph) ×2 + 1, nrows(Graph) ×2 + 1)
5  foreach row ∈ nrows(Graph) do
6      foreach col ∈ ncols(Graph) do
7          p ← col + row× ncols(Graph)
8          array(grid,row × 2 + 1,col × 2 + 1) ← 0
9          foreach adjacent ∈ Adjacency(A) do
10             qCol ← col+ dx(A,index)
11             qRow ← row+ dy(A,index)
12             if (qCol ≥ 0 & qCol < ncols(Graph)) ;
13             & (qRow ≥ 0 & qRow < nRows(Graph)) then
14                 weight ← link(Graph,p,index)
15             else
16                 weight ← 0
17             if col = qCol then
18                 array(grid,row × 2 + 1,min(col, qCol) ×2 + 2) ← weight
19             else if row = qRow then
20                 array(grid,min(row, qRow) ×2 + 2,col × 2 + 1) ← weight
21 foreach row ∈ nrows(Graph) do
22     foreach col ∈ ncols(Graph) do
23         weight ← 0 foreach adjacent ∈ Adjacent(A) do
24             qCol ← col × 2+ dx(A, adjacent)
25             qRow ← row × 2+ dy(A, adjacent)
26             if qCol ≥ 0qCol < ncols(grid) qRow ≥ 0qRow <
                nrows(grid) then
27                 weight ← max(weight,array(grid,qRow,qCol))
28         array(grid,row × 2,col × 2) ← weight
29 return grid
```

4 Results

In this section, we show examples of our proposed manipulation method of the UOIFT hierarchy on both grayscale and colored images. For every image, we also show its superpixel segmentation by IFT-SLIC as used by UOIFT, for better visualization of all the intermediate results. In all examples presented in this section, the Khalimsky Grid of saliency maps used for display purposes of hierarchies will be denoted simply as saliency maps for short.

The original hierarchy by UOIFT is compared to the proposed volume-based rearranged hierarchy in the task of isolating objects of interest with the fewest possible partitions of the image. As a result, for all examples, the new approach was able to reduce the number of partitions needed to isolate the objects in the images compared to UOIFT, which generates many small regions of small artifacts (see Figures 10, 11, 12, 13, 14, 15 and 16).

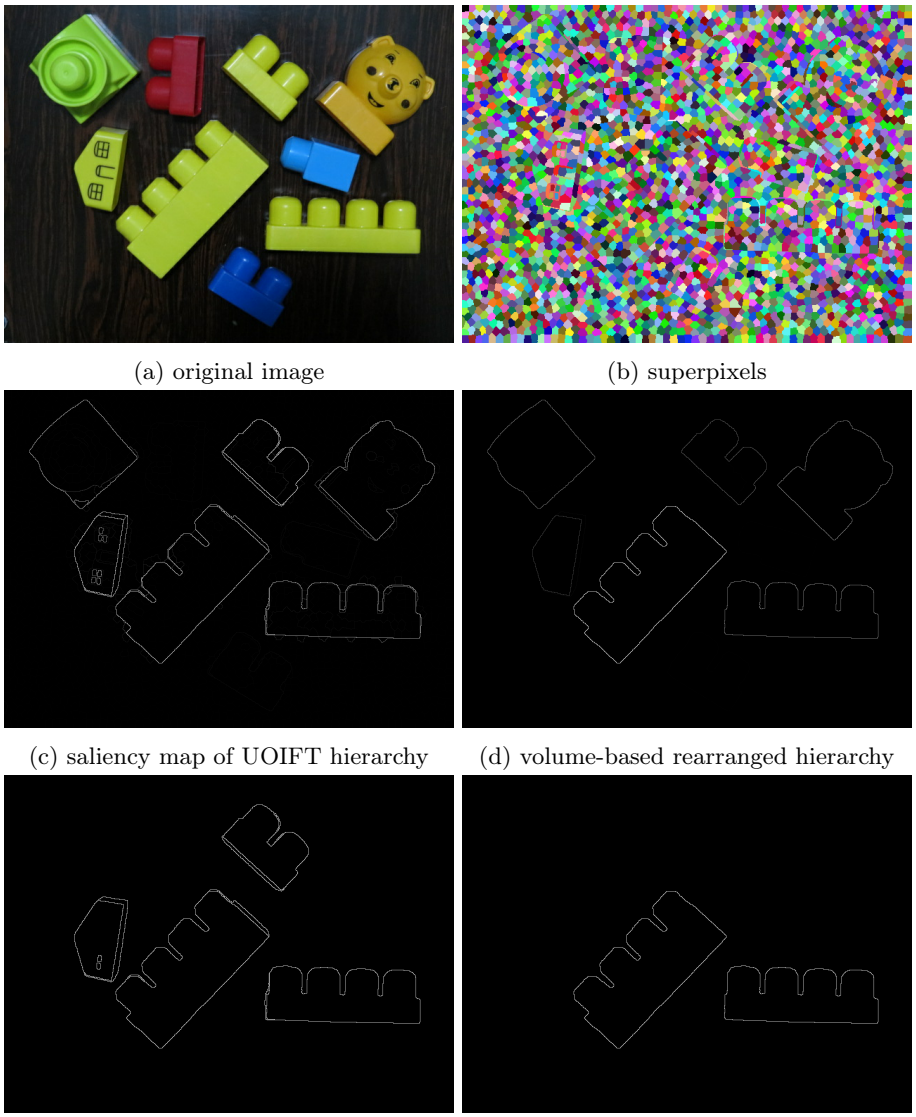


(a) original image (b) superpixels

(c) saliency map of UOIFT hierarchy (d) volume-based rearranged hierarchy

(e) segmentation by thresholding of "c" (f) segmentation by thresholding of "d"

Figure 10: (a) Example of a synthetic image where the object of interest is the larger central gray circle. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring transitions from bright to dark pixels. (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires five regions to segment the desired object. (f) An improved result, requiring only two regions, is obtained by thresholding the rearranged hierarchy in "d".



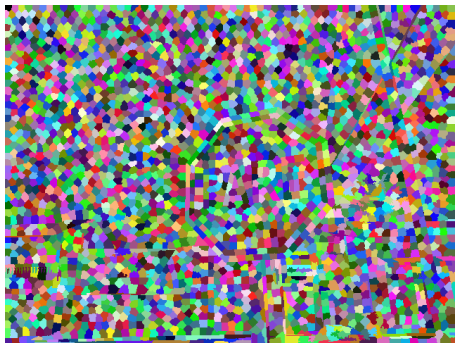
(a) original image (b) superpixels

(c) saliency map of UOIFT hierarchy (d) volume-based rearranged hierarchy

(e) segmentation by thresholding of "c" (f) segmentation by thresholding of "d"

Figure 11: (a) Example of an image containing parts of a building toy where we are interested in segmenting the two largest yellow blocks. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring color transitions from hexadecimal RGB code #a9c53c (yellow sample) to #26231e (background sample). (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than ten regions to segment the desired objects. (f) An improved result, requiring only three regions, is obtained by thresholding the rearranged hierarchy in "d".

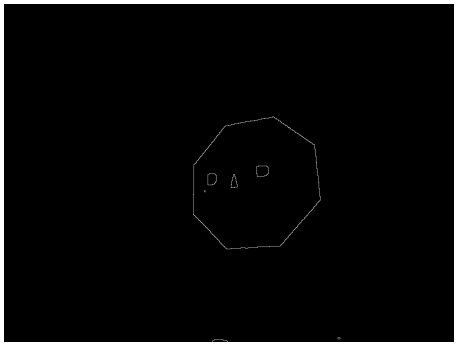
(a) original image



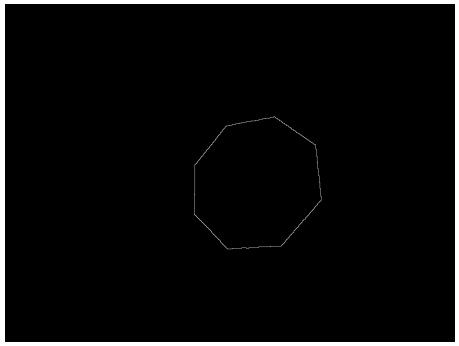
(b) superpixels

(c) saliency map of UOIFT hierarchy

(d) volume-based rearranged hierarchy

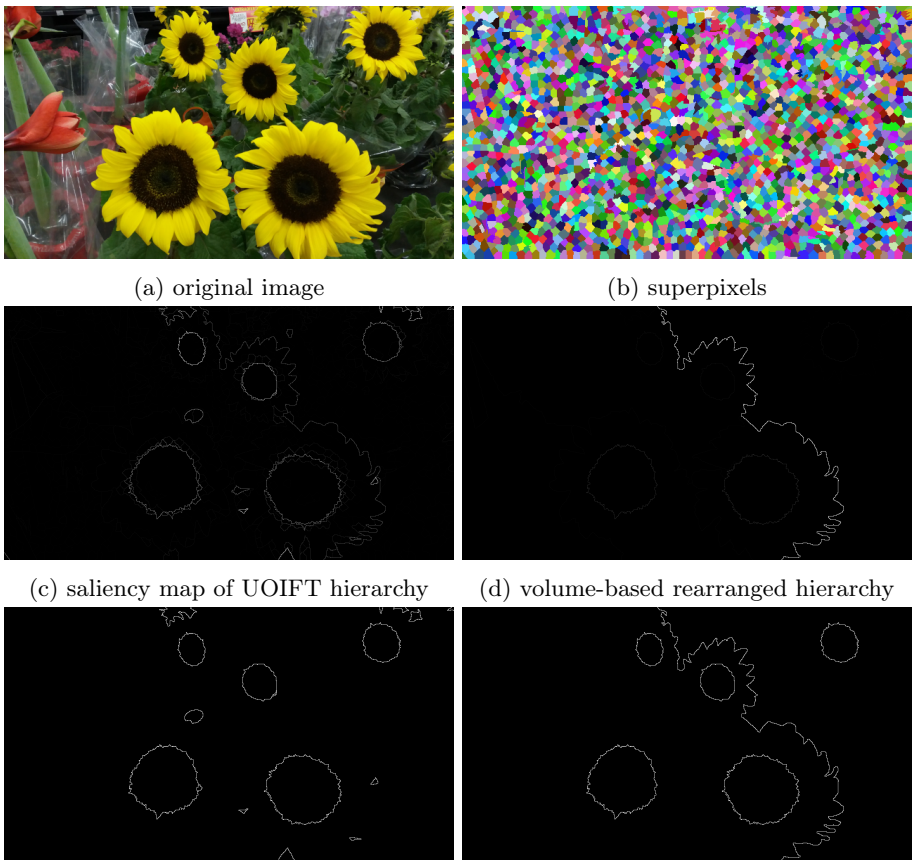


(e) segmentation by thresholding of "c"



(f) segmentation by thresholding of "d"

Figure 12: (a) Example of a STOP traffic signal where we are interested in segmenting its internal plate contour. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring color transitions from hexadecimal RGB code #9a1310 (red sample) to #8f8677 (white sample). (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than seven regions to segment the desired object. (f) An improved result, requiring only two regions, is obtained by thresholding the rearranged hierarchy in "d".

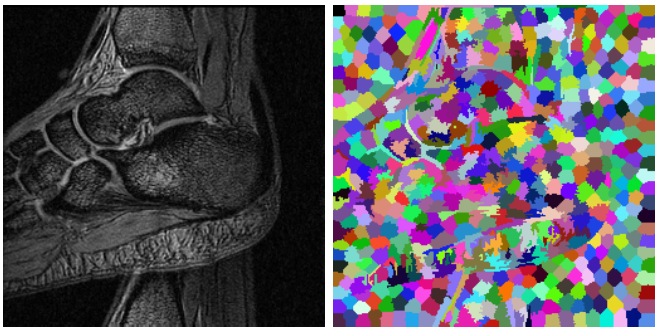


(a) original image (b) superpixels

(c) saliency map of UOIFT hierarchy (d) volume-based rearranged hierarchy

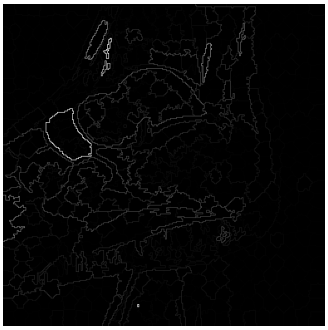
(e) segmentation by thresholding of "c" (f) segmentation by thresholding of "d"

Figure 13: (a) Example of sunflowers image where we are interested in segmenting their seeds (dark part) without the petals (yellow part). (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring color transitions from hexadecimal RGB code #1f1509 (sample color at seeds) to #c9b300 (sample color at petals). (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than sixteen regions to segment the desired objects. (f) An improved result, requiring only seven regions, is obtained by thresholding the rearranged hierarchy in "d".

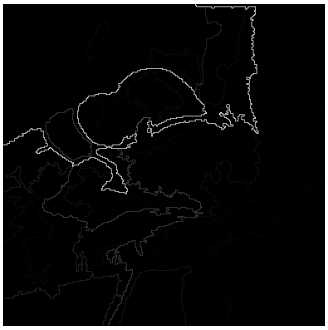


(a) original image

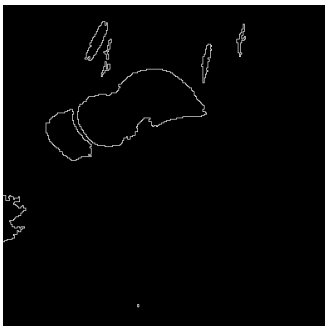
(b) superpixels



(c) saliency map of UOIFT hierarchy



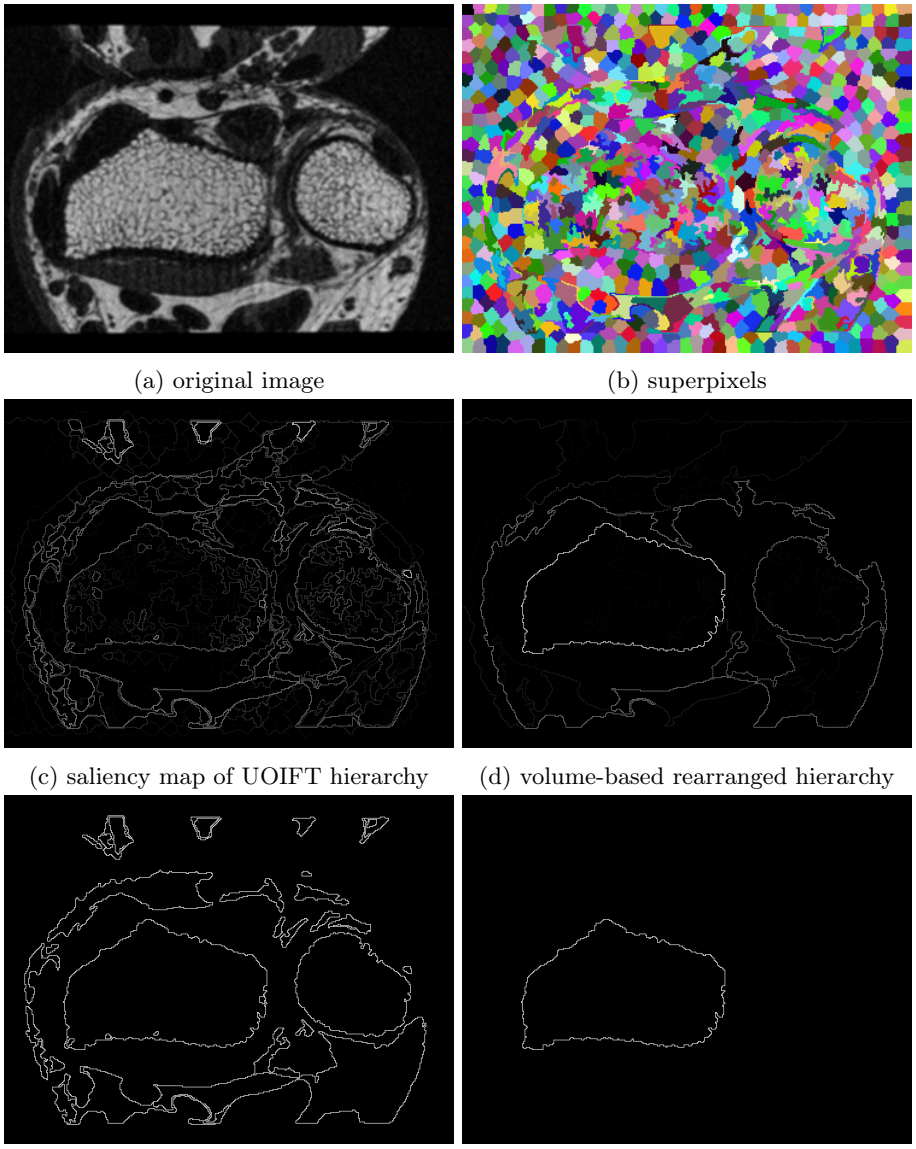
(d) volume-based rearranged hierarchy



(e) segmentation by thresholding of "c"

(f) segmentation by thresholding of "d"

Figure 14: (a) Example of an MR image of the foot where the desired object is the talus bone. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring transitions from dark to bright pixels. (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than nine regions to segment the desired object. (f) An improved result, requiring only three regions, is obtained by thresholding the rearranged hierarchy in "d".

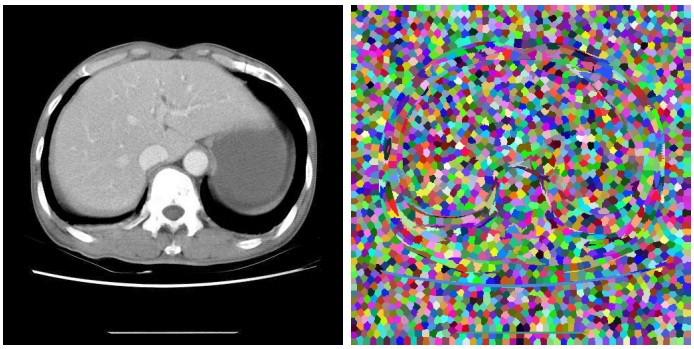


(a) original image (b) superpixels

(c) saliency map of UOIFT hierarchy (d) volume-based rearranged hierarchy

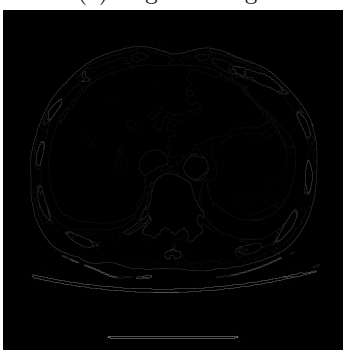
(e) segmentation by thresholding of "c" (f) segmentation by thresholding of "d"

Figure 15: (a) Example of an MRI slice of the wrist. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring transitions from bright to dark pixels. (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than twenty regions to segment the wrist. (f) An improved result, requiring only two regions, is obtained by thresholding the rearranged hierarchy in "d".

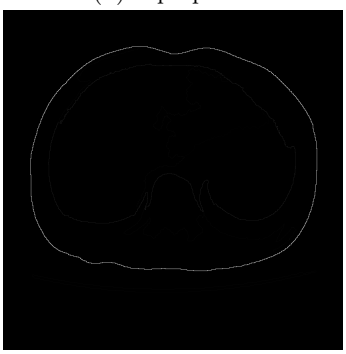


(a) original image

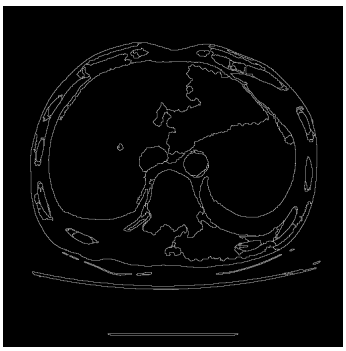
(b) superpixels



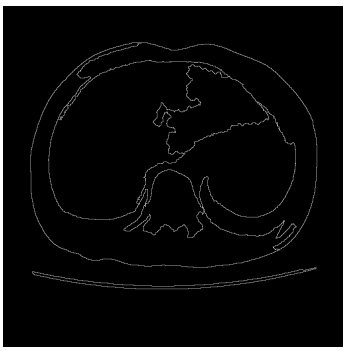
(c) saliency map of UOIFT hierarchy



(d) volume-based rearranged hierarchy



(e) segmentation by thresholding of "c"



(f) segmentation by thresholding of "d"

Figure 16: (a) Example of a thoracic CT image of 512×512 pixels to segment the liver. (b) Superpixels by IFT-SLIC. (c) The saliency map of the hierarchical image segmentation by UOIFT, favoring transitions from bright to dark pixels. (d) The saliency map of the volume-based rearranged hierarchy. (e) The best image partition by thresholding the saliency map in "c" requires more than forty regions to segment the liver. (f) An improved result, requiring only eight regions, is obtained by thresholding the rearranged hierarchy in "d".

5 Conclusion

Throughout this work, we've shown the potential and results of the new proposed UOIFT [2] and Min-tree combination for image segmentation. Looking at the differences between our method and the one shown in [2], by means of side-by-side comparisons on the last section, it is clear how the pure UOIFT hierarchy produces partitions with smaller regions, being consequently harder to achieve an automatic segmentation of the most prominent regions of the image. In the proposed method, on the other hand, we were able to successfully combine the size information of the objects with their expected boundary polarity leading to improved results.

In this work, the code implementation of the methods was done in C++, due to it's superior performance and memory management capabilities when compared to other, higher level programming languages. Consequently, some of the packages used in the development of required data structures limited its portability, unfortunately being restricted to GNU/Linux environments.

As future works, other ideas stemming from recent works of this area, such as [4], [3] and [8], could also be explored.

6 Challenges

6.1 Covid-19 and the pandemic

As with many services and projects throughout society, this work was also affected by the global pandemic. During the weeks of development, as both the state and city governments applied successive restrictions on local citizens (including, but not limited to, limitations on personnel and available activities in our work space at University of São Paulo), communication between the researchers responsible was severely hindered, due to the new and substantial demand on the traditional channels adopted by the team. Fortunately, everyone working on the project was able to adapt to a remote development setup, with Google Meet as the primary mean of video communication.

6.2 Hierarchies and Max-trees

Understanding the concept of an image hierarchy and its ties to Khalimsky grids was also a challenge. Not only that, but also having to comprehend quasi-flat zones at the same time, which were reasonably new concepts for me, certainly took an unexpected amount of time.

Combining these new paradigms with Max-trees was, by far, the biggest hurdle in development. While Khalimsky grids utilize the edge weights to build new image graphs, Max-trees are designed around node attributes, so the translation between these two data structures was not trivial.

Also, it is essential to take into account that a node in the Max-tree is a superpixel, which represents a group of pixels (nodes) in the image graph that generates it, making the linkage between pixels in the image and Max-tree not trivial. It is for this problem that the $inverseMap$ array was utilized in Algorithms 4 & 5.

6.3 C/C++ language

Due to the nature of this work, that is, processing images in a reasonably small time frame, it quickly became clear that a low-level environment was needed to suffice our performance requirements. Therefore, C/C++ was a simple choice for development.

As with any other programming language, both C & C++ were challenging to work at first. Specifically, their memory allocation aspect, absent in most high-level languages commonly utilized today, was the most difficult feature to work within the C/C++ programming environment. The infamous *segmentation fault* errors were a familiar sight during development, but, fortunately, we were able to overcome such obstacles and craft fine, working software.

References

[1] E. B. Alexandre, A. S. Chowdhury, A. X. Falcao, and P. A. V. Miranda. IFT-SLIC: A general framework for superpixel generation based on simple linear iterative clustering and image foresting transform. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 337–344, 2015.

[2] Hans H. C. Bejar, Silvio Jamil Ferzoli Guimarães, and Paulo A. V. Miranda. Efficient hierarchical graph partitioning for image segmentation by optimum oriented cuts. *Pattern Recognition Letters*, 131:185–192, 2020.

[3] Isabela Borlido, Gabriel B Fonseca, Zenilton Patrocinio Jr, Jean Cousty, Benjamin Perret, Laurent Najman, Yukiko Kenmochi, and Silvio Guimarães. Exploring hierarchy simplification for non-significant region removal. *XXXII Conference on Graphics, Patterns and Images.*, 2020.

[4] Edward Cayllahua-Cahuina, Jean Cousty, Silvio Jamil F Guimarães, Yukiko Kenmochi, Guillermo Cámara-Chávez, and Arnaldo de Albuquerque Araújo. Hierarchical segmentation from a non-increasing edge observation attribute. *Pattern Recognition Letters*, 131:105–112, 2020.

[5] Jean Cousty, Laurent Najman, Yukiko Kenmochi, and Silvio Guimarães. Hierarchical segmentations with graphs: Quasi-flat zones, minimum spanning trees, and saliency maps. *Journal of Mathematical Imaging and Vision*, 60:479–502, 2018.

[6] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59:167–181, 2004.

[7] Silvio Guimarães, Yukiko Kenmochi, Jean Cousty, Zenilton Patrocinio, and Laurent Najman. Hierarchizing graph-based image segmentation algorithms relying on region dissimilarity: the case of the felzenszwalb-huttenlocher method. *Mathematical Morphology - Theory and Applications*, 2:55–75, 2017.

[8] B Perret, J Cousty, SJF Guimarães, Y Kenmochi, and L Najman. Removing non-significant regions in hierarchical clustering and segmentation. *Pattern Recognition Letters*, 128:433–439, 2019.

[9] P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Transactions on Image Processing*, 7:555–570, 1998.

[10] Alexandre Gonçalves Silva and Roberto de Alencar Lotufo. Efficient computation of new extinction values from extended component tree. *Pattern Recognition Letters*, 32:79–90, 2011.

[11] Alexandre Gonçalves Silva and Roberto de Alencar Lotufo. Efficient computation of new extinction values from extended component tree. *Pattern Recognition Letters*, 32:79–90, 2011.