Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: {mcerman}@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-133 September 9, 2015

Structurally Correct Image Segmentation using Local Binary Patterns and the Combinatorial Pyramid

Martin Cerman

Abstract

In this work we present a new image segmentation algorithm which is based on Local Binary Patterns and the Combinatorial Pyramid. Current Local Binary Pattern-based segmentation algorithms utilize statistical approaches in form of a histogram to describe and compare textured regions, and to subdivide an image into homogeneous regions. The novelty of our approach is that we omit the usage of histograms and perform a segmentation based directly on the local structure of the image, while at the same time preserving structural correctness and image topology. In our work we define five topological classes that are based on the Local Binary Patterns of regions and are invariant to the number and shifting of bits, namely local minima, slopes, singular slopes, saddles, and local maxima. Using these classes in combination with the dual graph we are able to identify and remove redundant structural information. This approach simplifies the image graph and enables a merging of connected regions without introducing structural errors. We compare our algorithm to five other algorithms using the Global Consistency Error and Probabilistic Rand Index error metrics. One of these algorithms is a pre-version of our proposed algorithm which does not take structural constraints into consideration, and the remaining four algorithms are existing algorithms based on internal- and external contrast, Minimum Spanning Trees, Mean-Shift, and superpixel approaches. The evaluation shows, that the proposed algorithm indicates comparably good results with the Global Consistency Error metric, and it beats all of the five algorithms in terms of a high Probability Rand Index score. This segmentation behavior suggests, that a refinement of segmentations takes place at regions where there is evidence of multiple levels of granularity of segmentations performed by human subjects, and thus an application in image compression can be found.

Acknowledgements

I would like to express my sincere gratitude to the main advisor of my Master's thesis, Prof. Walter G. Kropatsch. Thank you for showing me the right direction when it was most needed. Without your valuable feedback, immense knowledge, patience, and guidance, my work would not be what it is now. Also thank you for introducing me to the world of research, and the opportunity to be an active part of it.

My gratitude also goes to my secondary advisor, Dr. Yll Haxhimusa. Thank you for supporting and mentoring me throughout the long years of my studies, sharing your ideas and opinions with me, and for your friendship.

Further, I would like to thank the PRIP Club, the organization of friends and promoters of Pattern Recognition and Image Processing activities, for the financial support, which enabled me to participate in international conferences and present my research.

Last but not least, I would like to thank my family and friends for their unconditional support throughout my life. Thanks to you I am who I am.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Image segmentation is an important part of Computer Vision. It refers to the subdivision of an image into non-overlapping heterogeneous regions, where each region consists of a set of pixels that share some homogeneous properties. These properties can be based on color, texture, or any other similarity criterion [FMR+02]. The idea is, that a good segmentation can capture perceptually important regions, which reflect local and/or global properties of the image [PZZ13]. These regions can then be used for classification and subsequently for higher level tasks such as image understanding, or detection and tracking of objects located in these regions. Over the last few decades hundreds of segmentation algorithms [FMR+02] have been proposed, which can be classified by their functionality, namely based on thresholding, histograms, edge detection, region growing and splitting/merging, Watershed Transformation, or graph partitioning [RRDR09].

Graph theoretical approaches model the image that is being segmented as a weighted graph. Each pixel is represented by a vertex in the graph, and the edges between the vertices represent adjacency relationships. Graph based segmentation methods can be categorized as Minimum Spanning Tree based, graph cut methods with cost functions and Markov random fields, shortest path based methods, and others [PZZ13]. Since the introduction of Local Binary Patterns in 1996, efforts have been made to integrate this local texture descriptor in image segmentation algorithms as well. Ojala et. al.[OP99] proposed an unsupervised three phase algorithm. Here the image is firstly hierarchically split into blocks based on an uniformity test involving the LBP, then agglomerative merging of the blocks is applied, and finally to get a high precision segmentation at the borders of individual regions, pixel-wise classification is used. Later in 2008, this algorithm was extended to video sequences by Chen et. al. [CZP08]. In 2006, Heikkilä et. al. [HP06] proposed a background segmentation algorithm which detects moving objects in a video sequence. This segmentation is reached by modeling the history of a region around a pixel over time as a group of weighted LBP histograms and comparing the current histogram to the stored samples. If the current histogram does not fit a

similarity criterion, the pixel is classified as foreground. In 2009, Chen et. al. [CZP09] further improved their texture segmentation algorithm for video sequences by combining the LBP texture descriptor with the WLD descriptor [CSH+10] which is based on the ratio of the intensity of a pixel and the relative intensity difference of its 8-neighbors. One common property of these mentioned approaches is, that they utilize a statistical description of the underlying texture using a histogram. The main disadvantage of this method however is that the spatial distribution of the local structures is not captured, and thus multiple images that are for us humans perceptually completely different, may be statistically equivalent. Based on our research and to our current knowledge there are no other methods, which would combine the advantages of the Local Binary Patterns with a graph based image segmentation approach without utilizing histograms.

In this work we propose a new image segmentation algorithm which is based on Local Binary Patterns without the use of histograms. Our algorithm represents an image as a pair of graphs that is stored using the combinatorial pyramid representation. The Local Binary Patterns of each vertex in the primal and dual graph can be divided into one of five topological classes, which are invariant to the number and shifting of the bits. Using these classes it is possible to transform the dual graph in such a way, that it consists, with the exception of the outer face, only of topological slopes. This way each face in the primal graph is enclosed by two directed paths. By merging two regions and analyzing the topological class of the incident faces it is possible to identify and remove structural information. This approach iteratively reduces the image graph up to the apex of the pyramid, while preserving the structural correctness of the image. As a consequence, lowly textured regions are merged early in the segmentation hierarchy, and highly textured regions are merged late. Perceptually this behavior causes important visual information for humans to be preserved even at a high level of region compression.

My contributions in this work are:

- Proposal of an image segmentation algorithm based on Local Binary Patterns which does not use a histogram to describe a region,
- Definition of structurally correct image segmentation and proposal of a method to ensure structurally correct image segmentation,
- Definition of topological classes that are invariant to the number and shifting of bits of a Local Binary Pattern,
- Identification and removal of redundant structural information,
- Proposal of a method for transforming all dual vertices (except the outer face) into slopes, and
- Evaluation, comparison to other algorithms, and analysis of the properties and applications for the proposed algorithm.

This work is divided as follows: Chapter 2 introduces the combinatorial map representation, and how the dual graph contraction theory is implemented using removal- and contraction operations in a combinatorial pyramid framework. Additionally a canonical representation of the combinatorial pyramid and its advantages, as well as the topology preserving properties of the combinatorial pyramid are presented in this chapter as well. The theory of Local Binary Patterns, and topological region classes deduced from these structural relationships are described in Chapter 3. This chapter also includes a method to assign strict inequality relationships onto the edges of the image graph, and a discussion about the special case of self-loops. The definition of structural equality and the computation of the representative image for the whole class of images with the same structure is described in Chapter 4. A discussion about structural redundancy, removal of dual saddles, as well as a connection to the order theory is also included in this chapter. Chapter 5 describes our proposed image segmentation algorithm, which is evaluated and compared to five other image segmentation algorithms in Chapter 6. A discussion regarding the results can also be found in that chapter. Conclusions about the proposed algorithm and its performance are drawn in Chapter 7. Plots with the exact segmentation results and the error curves used in the evaluation can be found in Appendix A.

CHAPTER 2

Combinatorial Pyramid

Regular image pyramids were introduced by Burt et al. [BHR81] in 1981 in the field of image segmentation. Regular pyramids are defined as a sequence of copies of an original image in which the resolution is decreased in regular steps. The relationships between regions in adjacent levels of the pyramid are fixed, which means that for each region in a higher level there is a fixed number of regions in the lower level connected by a fixed relation. A reduction window relates each region in the pyramid with a set of regions defined in the level below. Therefore a regular image pyramid is defined by a fixed reduction window and a fixed reduction factor [BK01]. Since their introduction, regular pyramids have been utilized in many other applications, such as data compression, multi-scale texture analysis, shape analysis, or motion analysis [AAB+84, Ros84].

Due to the lack of flexibility of the regular pyramids, irregular pyramids [MMR91] were introduced. These are defined as a stack of successively reduced graphs, where each graph is built from the graph below by selecting a specific subset of (surviving) vertices and mapping the remaining ones onto these vertices. This way the reduction factor between adjacent levels is not fixed, and the size of each level and the height of the pyramid are unknown prior to generation [MR06]. It was however shown that the height of the irregular pyramid can be bounded by a logarithmic height [HGS+02]. The main advantage of irregular pyramids is, that a vertex in the higher levels of the pyramid can represent various shapes of connected regions in the pixel level. These irregular regions can represent objects in an image, which can further be recognized, analyzed or tracked. Another advantage is that the merging hierarchy of the vertices in the pyramid is captured as well. An example of an irregular pyramid can be seen in Figure 2.1. Irregular pyramids have been successfully used in tasks such as image segmentation [HK04, GHK11], text segmentation and binarization [LT03], object tracking [MR06].

Irregular pyramids can be stored in several data structures [MR06], namely as simple region adjacency graphs, dual graphs, combinatorial maps or generalized maps. Simple

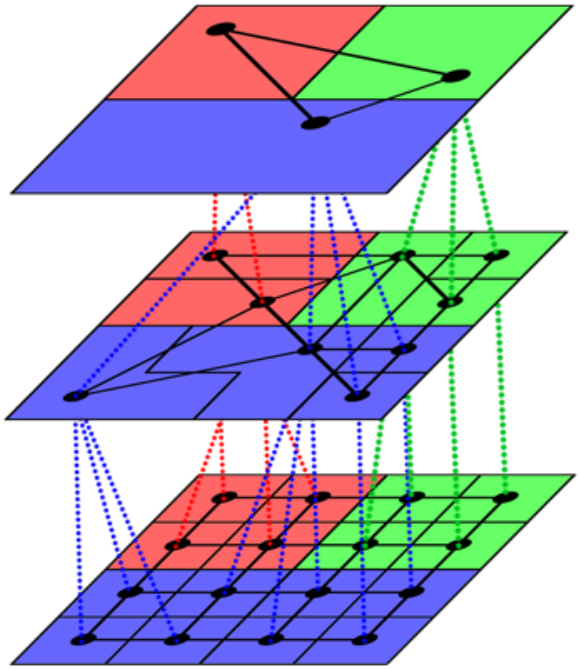


Figure 2.1: An example of an irregular pyramid.

graphs have the disadvantage, that the number of common boundaries between two adjacent regions is not known, as well as they do not allow to differentiate an adjacency relationship between two regions from an inclusion relationship. Since the neighborhood relationships of regions in an image produce a planar graph, it is possible to define dual graphs. This way the drawbacks with the simple graph data structure are removed, but at a cost of increased storage requirements. A combinatorial map may be considered as a planar graph with an explicit encoding of the neighborhood's orientation of each vertex and an implicit encoding of the dual graph [BK01]. The combinatorial map also encodes multiple boundaries and inclusions, and this way preserves the topology of the image [Hax07, KHI07]. A generalized map is an extension of the combinatorial map, which additionally allows representation of non-orientable surfaces and open subdivisions [GSDL06]. As these properties of the generalized map are not needed in this work, the representation for storing each level of the irregular pyramid will be a combinatorial map. This stack of combinatorial maps will form a combinatorial pyramid [BK01].

This chapter is organized as follows: Section 2.1 defines the combinatorial map, the dual combinatorial map, and special edges which need to be identified when applying reduction operations. Section 2.2 introduces the combinatorial pyramid, the theory of dual graph- and combinatorial map reduction, as well as a canonical representation of the combinatorial pyramid. Finally Section 2.3 discusses the topology preserving properties of the combinatorial pyramid and presents some examples, and Section 2.4 shows the relation between dual graphs and a combinatorial map.

2.1 Combinatorial Map

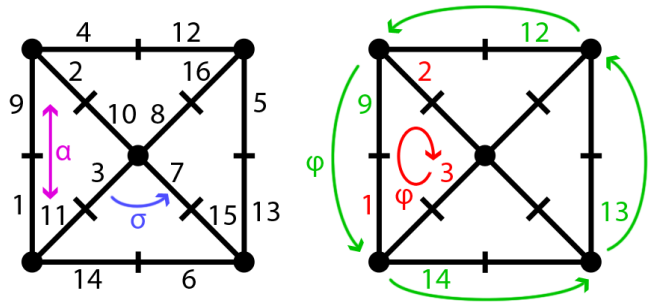
An n-dimensional combinatorial map is a combinatorial representation allowing to describe a n-dimensional oriented quasi-manifold with or without boundary [DL14]. Figuratively speaking, a n-dimensional oriented quasi-manifold is an object obtained by "gluing" together its n-cells along its $(n-1)$-cells, and when embedding it in an Euclidean space it is possible to define at each point of this object an unambiguous direction. In the case of a 2-dimensional object 0-cells are its vertices, 1-cells are its edges, and 2-cells are its faces. An image graph is such a 2-dimensional object and the individual cells correspond to vertices representing pixels, edges representing the neighborhood relationship between the pixels, and faces representing 2×2-configurations of pixels. Our work is based on 2-dimensional combinatorial maps, which will be just called combinatorial maps for the remainder of this work.

Definition 1. Combinatorial Map
A combinatorial map is a triplet $C = (\mathcal{D}, \alpha, \sigma)$, where:

- *$\mathcal{D}$ is a finite set of darts*,
- *α is an involution on the set $\mathcal{D}$*, and
- *σ is a permutation on the set $\mathcal{D}$.*

Compared to a simple graph data structure, an image is not encoded as a set of vertices and edges, but as a set of darts $\mathcal{D}$. Each edge in the graph is represented as two half edges called *darts*, d_1 and d_2, connected by the involution $\alpha(d_1) = d_2$ and $\alpha(d_2) = d_1$. As a result the relationship $\alpha(\alpha(d)) = d$ holds for all darts in the combinatorial map. This encoding of an edge in the combinatorial map is also called a (alpha)-*orbit* and is denoted by $\alpha^*(d)$. Additionally each dart is associated with a vertex in the graph. The σ permutation relates each dart with the following dart around the same vertex in clockwise- or counter-clockwise direction. The direction of the encoding is implementation specific. The sequence of darts encountered when turning around a vertex starting from dart d is called a (sigma)-*orbit* and is denoted by $\sigma^*(d)$.

A combinatorial map implicitly encodes a dual graph [Har69], which can be constructed using a combination of the α involution and the σ permutation. The encoding of the edges stays the same, namely using the α involution, but the encoding of the vertices changes. The orbits of the dual permutation φ encode the set of darts when turning around the vertices of the dual graph. Depending on the chosen direction of the σ-orbit, the dual permutation φ has the opposite direction and is defined as $\varphi = \alpha \circ \sigma$ or $\varphi = \sigma \circ \alpha$. In the primal graph the orbit $\varphi^*(d)$ returns the sequence of darts when turning around a face. This way, the dual $\overline{C}$ of a combinatorial map $C = (\mathcal{D}, \alpha, \sigma)$ is defined as $\overline{C} = (\mathcal{D}, \alpha, \varphi)$. An example of a simple graph encoded in a combinatorial map data structure is shown in Figure 2.2. Notice in this figure, that the background face (green color) is encoded in



$\mathcal{D}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
α	9	10	11	12	13	14	15	16	1	2	3	4	5	6	7	8
σ	14	4	7	9	12	13	8	10	2	3	1	16	15	11	6	5
φ	2	3	1	16	15	11	6	5	14	4	7	9	12	13	8	10
φ'	6	12	15	1	4	5	16	2	10	11	9	8	7	3	14	13

Figure 2.2: An example of a simple graph encoded as a combinatorial map.

the opposite direction compared to the inner face (red color). This is a phenomenon of the dual combinatorial pyramid $\overline{C}$ and is described in detail in [Bru02]. The last two rows of the table in the figure are both possible φ permutations, namely the first being $\varphi = \alpha \circ \sigma$ and the second being $\varphi' = \sigma \circ \alpha$.
As will be seen in the next section, some special types of edges in a graph can cause graph connectivity problems when two vertices are being merged or edges are being removed. Using a combinatorial map encoding, these edges can however easily be detected. Brun and Kropatsch [BK99] defined these edges as follows:

Definition 2. Empty Self-loop
An edge $\alpha^*(d)$ is an empty self-loop, iff $\sigma(d) = \alpha(d)$.

Definition 3. Self-loop
An edge $\alpha^*(d)$ is a self-loop, iff $\alpha(d) \in \sigma^*(d)$. An empty self-loop always fulfills this condition.

Definition 4. Pendant Dart
A dart d is called a pendant (or dangling) dart, iff $\sigma(d) = d$. In this case the vertex $\sigma^*(d) = (d)$ is called a pendant vertex.

Definition 5. Pendant Edge
An edge $\alpha^*(d)$ is called a pendant (or dangling) edge, iff d or $\alpha(d)$ are a pendant dart.

Definition 6. Bridge
An edge $\alpha^*(d)$ is called a bridge, iff $\alpha(d) \in \varphi^*(d)$.

Already on the special case of a bridge the usefulness of switching between a combinatorial map and its dual can be observed. As it is mentioned in [BK99], this relationship between the primal and dual combinatorial map reaches even deeper. One common property for example is that when the primal combinatorial map is connected, its dual is connected as well. Moreover, the previously defined special edges are related as well [BK01]:

"primal map" $(\mathcal{D}, \alpha, \sigma)$		*"dual map"* $(\mathcal{D}, \alpha, \varphi)$
self-loop	$\alpha(d) \in \sigma^*(d)$	bridge
bridge	$\alpha(d) \in \varphi^*(d)$	self-loop
empty self-loop	$(\sigma(d) = \alpha(d)$ or $\sigma(\alpha(d)) = d)$	pendant dart
pendant dart	$(\sigma(d) = d$ or $\sigma(\alpha(d)) = \alpha(d)$	empty self-loop

Table 2.1: Relationships between special types of edges in the primal and dual combinatorial map.

2.2 Combinatorial Pyramid

A combinatorial pyramid is a stack of successively reduced combinatorial maps. At the base level of the pyramid is the initial combinatorial map, and at the higher levels are combinatorial maps produced by sequentially applying contraction and removal operations on the combinatorial maps at the lower levels. The contraction and removal operations were defined by Kropatsch [Kro98] as a reduction scheme for dual graphs. Before continuing with the contraction and removal operations and a canonical representation of the combinatorial map, we will briefly explain the reduction scheme of dual graphs.

2.2.1 Dual Graph Reduction Scheme

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. The reduction of the graph can be performed by either an edge contraction- or edge removal operation. When an *edge removal operation* is applied to a graph G, the resulting graph G' is a subgraph of G with $G' = (V, E \setminus \{e_0\})$, where e_0 is the removed edge. Similarly, the graph G' resulting from an *edge contraction operation* is a subgraph of G with $G' = (V \setminus \{v_0\}, E \setminus \{e_0\})$, where v_0 is one of the vertices incident to the contracted edge e_0. Let $G_k = (V_k, E_k)$ be a graph in a dual graph pyramid at level k and $G_{k+1} = (V_{k+1}, E_{k+1}) \subset G_k$ be at level $k+1$. Prior to performing a contraction operation, a subset $S_k = V_{k+1} \subset V_k$ of *surviving* vertices and a subset $N_{k,k+1} \subset E_k$ (where index k,$k+1$ mean the contraction from level k to $k+1$) of *non-surviving* edges is chosen. The surviving vertices $v \in S_k$ will survive to the next higher level of the pyramid, making up the set of vertices V_{k+1} of graph G_{k+1}, and the set of *non-surviving* vertices $v \in V_k \setminus S_k$ will be contracted towards their neighboring surviving vertices. To reduce the graph G_k it is necessary to satisfy two conditions, namely the graph (V_k, N_k) has to be a spanning forest of graph $G_k = (V_k, E_k)$, and the surviving vertices $v \in S_k \subset V_k$ are the roots of the forest (V_k, N_k). This set of

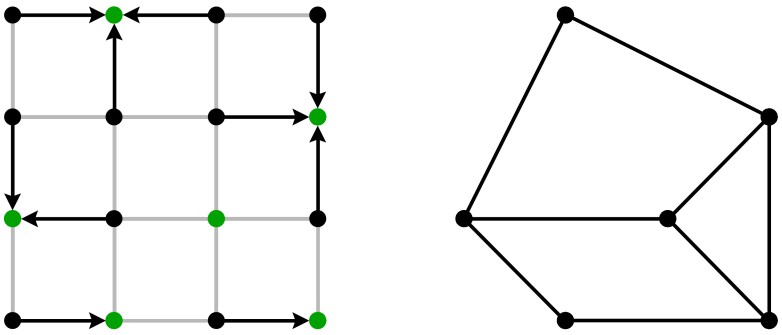


Figure 2.3: An example of contraction kernels in a graph. The left image shows the graph G_k, where the surviving vertices $v \in S_k$ are marked green, and the non-surviving edges $e \in N_k$ are marked black with an arrow pointing in the direction of contraction. The marked set of disjoint rooted trees is the set of contraction kernels. The right image shows the resulting graph G_{k+1} after dual graph contraction.

disjoint rooted trees with maximal length of two (from leaf over root to leaf) is called a set of *contraction kernels*. A detailed review of existing methods how to build these contraction kernels can be found in [Hax07]. An example of such contraction kernels can be seen in Figure 2.3.

Knowing all the contraction kernels from the base level to the level k, it is also possible to calculate directly the graph G_k. The individual contraction kernels for each level are stored as *decimation parameters* $(S_k, N_{k,k+1})$. The transition from base level to level k is done by combining successively the decimation parameters from level 0 to level k into one single *equivalent contraction kernel* (ECK). For example the graph G_{k+2} can be computed as [Kro98]:

$$C[C[G_k, (S_k, N_{k,k+1})], (S_{k+1}, N_{k+1,k+2})] = C[G_k, (S_{k+1}, N_{k,k+2})] = G_{k+2},$$

where C is a contraction operation. An example of equivalent contraction kernels of the same pyramid, but at different levels can be seen in Figure 2.4.

The removal- and contraction operations are very closely related in dual graphs, namely a removal operation in the primal graph is a contraction operation in its dual, and vice-versa. The dual graph reduction scheme proposed by Kropatsch [Kro98] applies first the contraction operation in the primal graph, called dual edge contraction, and successively a contraction operation in the dual graph, called dual face contraction. An example of this procedure can be seen in Figure 2.5. Here the first operation contracts non-survivors towards survivors, and if possible the second operation simplifies the graph by removing multiple edges and empty self loops. In the following section, this reduction scheme will be adapted to be usable with the combinatorial map representation. All of the properties of the graph dualism will be preserved.

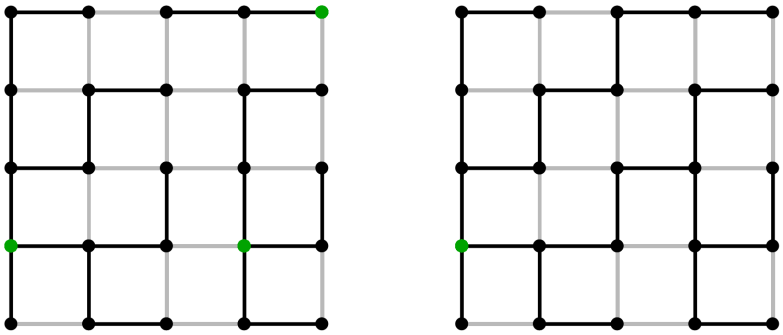


Figure 2.4: An example of equivalent contraction kernels on different levels of the same pyramid. The left image shows the ECK of G_k and the right image the ECK of G_{k+1}. The right ECK is a tree spanning over all vertices, which means that G_{k+1} is the apex of the pyramid.

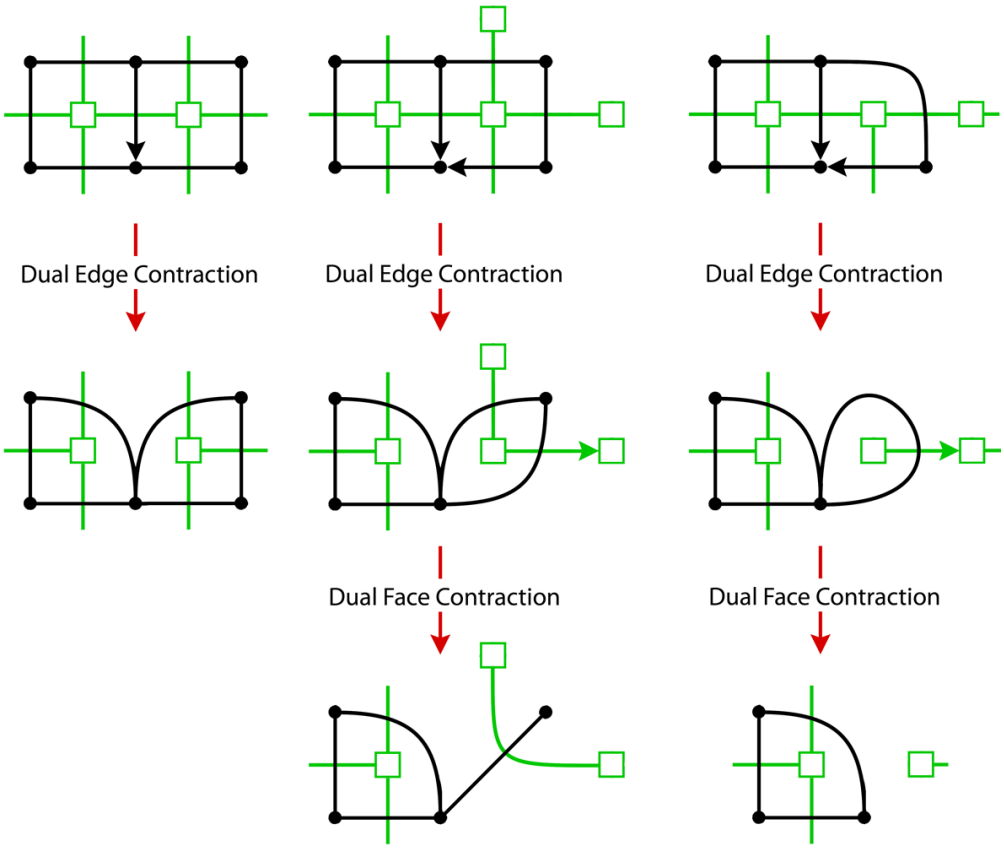


Figure 2.5: Examples of the dual graph reduction scheme. The primal graph is drawn black and its dual green. The middle and right examples show also the second simplification step by removing multiple edges and empty self loops using the dual face contraction.

2.2.2 Combinatorial Pyramid Reduction Scheme

In the combinatorial pyramid reduction scheme both the contraction- as well as the removal operation are performed by modifying the σ-permutation, and leaving the α-permutation untouched. This modification is visually easily understandable, namely the α-permutation binds together the two halves of an edge, and the σ-permutation binds together the half edges incident to a vertex in a chosen direction. When a contraction- or removal operation is applied, only the connections between vertices and edges are modified, not the edges themselves. Figure 2.6a shows a simple example of a removal of an edge connecting two vertices of degree four. After performing the removal operation, on both vertices the binding of the half edges is changed by skipping the removed half edges. In terms of a combinatorial map encoding this is a modification of the σ-permutation of the predecessor darts of the darts forming the removed edge. Mathematically the removal operation can be defined as [BK01]:

Definition 7. Removal Operation

Given a combinatorial map $C = (\mathcal{D}, \alpha, \sigma)$ and a dart $d \in \mathcal{D}$ which is neither a bridge nor an empty self-loop, the combinatorial map $C \setminus \alpha^*(d) = (\mathcal{D} \setminus \alpha^*(d), \alpha, \sigma')$ is defined by:

$$\begin{cases} \forall d' \in \mathcal{D} \setminus \sigma^{-1}(\alpha^*(d)) & \sigma'(d') = \sigma(d') \\ \sigma'(\sigma^{-1}(d)) = \sigma(d) \\ \sigma'(\sigma^{-1}(\alpha(d))) = \sigma(\alpha(d)) \end{cases}$$

Similarly, the contraction operation removes the edge between two vertices and merges them together. A visualization of the contraction operation can be seen in Figure 2.6b. Once again, this is only a modification of the half edge order of both vertices, namely the contracted edge is ignored and the order of half edges of both vertices is combined together. Same as with the removal operation, this only modifies the σ-predecessors of the darts forming the contracting edge in the combinatorial map.

Definition 8. Contraction Operation

Given a combinatorial map $C = (\mathcal{D}, \alpha, \sigma)$ and a dart $d \in \mathcal{D}$ which is neither a pendant edge nor a self-loop. The combinatorial map $C/\alpha^*(d) = (\mathcal{D} \setminus \alpha^*(d), \alpha, \sigma')$ is defined by:

$$\begin{cases} \forall d' \in \mathcal{D} \setminus \sigma^{-1}(\alpha^*(d)) & \sigma'(d') = \sigma(d') \\ \sigma'(\sigma^{-1}(d)) = \sigma(\alpha(d)) \\ \sigma'(\sigma^{-1}(\alpha(d))) = \sigma(d) \end{cases}$$

As it was explained in Section 2.1, there are special types of edges which can cause problems when contracting or removing an edge. For the contraction operation this special edge is a self-loop, and for the removal operation this is a bridge. The reasoning behind this is fairly obvious, namely one of the properties of the graph dualism is that when the

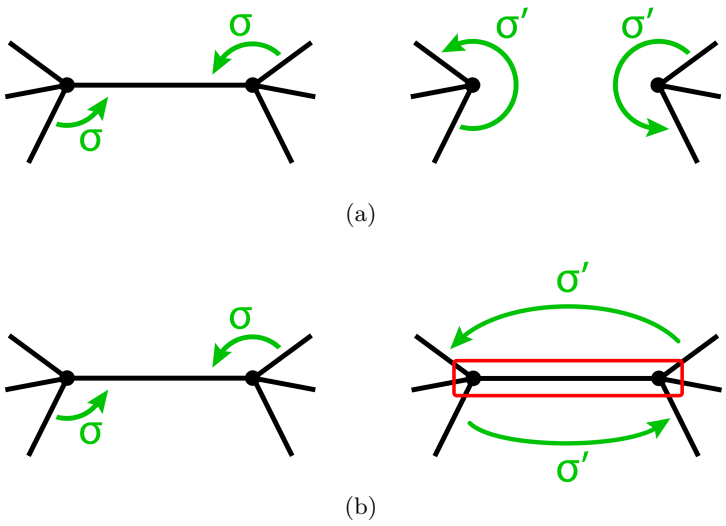


Figure 2.6: A visualization of the (a) removal operation, and the (b) contraction operation.

primal graph is connected, its dual is connected as well. When an image is represented as a planar graph and contraction- or removal operations are performed, then the connectivity of this image has to be preserved. In terms of a graph this means that edges that are bridges cannot be removed. In the dual graph this translates to a contraction of a self-loop.

The second step of the reduction scheme, namely simplification of the graph, can be performed very easily in the dual combinatorial map encoding. Multiple edges and empty self-loops can be identified in the dual combinatorial map by the cardinality of the φ-orbit [BK01]. An empty self-loop translates in the dual to a pending edge, which can be found by identifying pendant darts for which the condition $\varphi(d) = d$ is true. Here the cardinality of the φ-orbit is 1. Multiple edges between two vertices can be found in the dual by identifying darts, which belong to φ-orbits of the degree 2. Here the condition $\varphi^2(d) = \varphi(\varphi(d)) = d$ holds.

2.2.3 Canonical Representation

So far only the theory of combinatorial pyramids and the reduction scheme have been presented, but no implementation specific details were given. The combinatorial pyramid is a hierarchy of successively reduced combinatorial maps. Practically what this means is that with the height of the pyramid, the memory requirements to store the combinatorial pyramid are increased, as in each level a new combinatorial map has to be captured. Until recently this was the main disadvantage of the combinatorial pyramid representation. In response to this problem Torres and Kropatsch [TK14] proposed a canonical representation for the combinatorial pyramid which encodes the whole pyramid at the memory requirements of the combinatorial map at the base level. The main idea

of this encoding is that the combinatorial pyramid consists of a single array in which its elements are ordered with respect to the construction history of the pyramid.

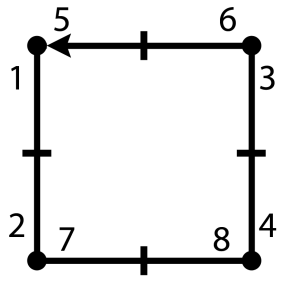
When an image is encoded as a combinatorial map, the number of darts equals to $|\mathcal{D}| = 2 * |E| = 2 * ((M-1) * N + M * (N-1))$, where $\mathcal{D}$ is the set of darts of the combinatorial map, E is the set of edges in the region adjacency graph of the image, and M and N are the image dimensions. The canonical encoding stores only the σ-permutation of the combinatorial map in an array of size $|\mathcal{D}|$. For the α-permutation, this encoding takes advantage of the indexing of the array. The darts are encoded in the α-permutation as even and odd unsigned integers, namely:

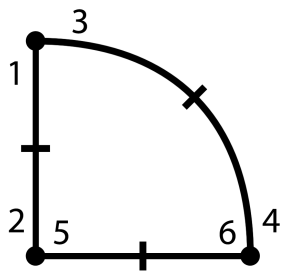
$$\alpha(d) = \begin{cases} d+1 & \text{if } d \text{ odd} \\ d-1 & \text{if } d \text{ even} \end{cases}$$

This encoding of the darts makes it redundant to store the labels of the darts and the α-permutation. The darts in the canonical encoding are numbered from 1 to $|\mathcal{D}|$. In a 0-indexed array this means for the σ-permutation, that for the dart d a look-up at position $d-1$ has to be made. Analogously, for a 1-indexed array the σ-permutation for dart d is stored at index d.

When a removal- or contraction operation on the base level of the pyramid is performed, the array is split into an active- and a passive part. The active part is stored as a whole in the front of the array and the passive part as a whole in the back of the array. This can be implemented by storing the index at which the passive part starts. The active part stores the state of the combinatorial map in the current level of the pyramid. When an operation on the current map is performed, a quadruple $(d, \alpha(d), \sigma(d), \sigma(\alpha(d)))$ defines this operation. The first two values, d and $\alpha(d)$, capture the indices at which an edge is encoded in the combinatorial map, and the latter two values, $\sigma(d)$ and $\sigma(\alpha(d))$, capture the encoding of that edge in the map. When an operation is performed, the σ-permutation of the affected darts is modified by following the reduction scheme, and then a reordering and relabeling of the darts takes place. This is done by moving the quadruple to the front of the passive part and relabel the darts to fit the corresponding position in the array. This procedure effectively stores the chronological order of the operations from the base level to the current level in the passive part of the array. An example of this reordering principle can be seen in Figure 2.7.

In order to retrieve the combinatorial map at the base of the pyramid, the darts in the passive part are moved to the active part. In this step there is no need for a reordering of the darts, because the darts are already chronologically ordered in the passive part of the pyramid. At this point, it is only necessary to apply the inverse removal- or contraction operations on the darts. Using the canonical encoding of the pyramid it is also possible to recognize whether an inverse removal or an inverse contraction operation should be





$\mathcal{D}$[i]	1	2	3	4	**5**	**6**	7	8
σ	5	7	6	8	**1**	**3**	2	4

$\mathcal{D}$[i]	1	2	3	4	5	6	**7**	**8**
σ	3	5	1	6	2	4	**1**	**3**

Figure 2.7: An example of an encoding of a combinatorial pyramid after a contraction operation. In the left combinatorial pyramid the quadruple defining the contraction is highlighted in gray. In the right pyramid, the quadruple moved to the passive part of the pyramid. The first rows of the tables are not stored in the array, as they are the dart labels defined by the indices of the array.

applied. The conditions for identifying an inverse removal operation are [TK14]:

- if $\sigma'(\alpha(d)) \notin \sigma'^*(\sigma'(d))$ or $\sigma'(d) \notin \sigma'^*(\sigma'(\alpha(d)))$,
- if $\sigma'^*(d) = \sigma'^*(\alpha(d))$,

where d and $\alpha(d)$ are the first two values of the quadruple identifying the operation at the beginning of the passive part. Similarly, the conditions for identifying an inverse contraction operation are:

- if $\sigma'(\alpha(d)) \in \sigma'^*(\sigma'(d))$ or $\sigma'(d) \in \sigma'^*(\sigma'(\alpha(d)))$.

The canonical representation of the combinatorial pyramid makes a trade-off between the space that is needed to store the pyramid along with the reduction history, and the computational cost that is needed to perform such a reduction operation. As it is mentioned in [TK14], the bits that are needed to store a combinatorial pyramid explicitly is $\log_2(\mathcal{D}) \sum_{l=0}^{n} \frac{|\mathcal{D}|}{k^l}$ plus parent/child relations to unfold the pyramid, where k^l is the reduction factor between level l and $l-1$, and n is the total number of levels. When an implicit encoding is used as in [BK03a, FB10], then the number of bits needed to store the pyramid is $|\mathcal{D}|\log_2(\mathcal{D}) + \frac{1}{2}|\mathcal{D}|(\log_2(n))$. The canonical representation of the pyramid reduces these space costs to the number of bits that are required to store the base combinatorial map of the pyramid, that is $|\mathcal{D}| \times \log_2(\mathcal{D})$.

With respect to the computational costs, it is also worth noting that when multiple reduction operations are performed on a single level of the pyramid, the reordering of the darts has to be applied only once after the last operation on that level has been performed.

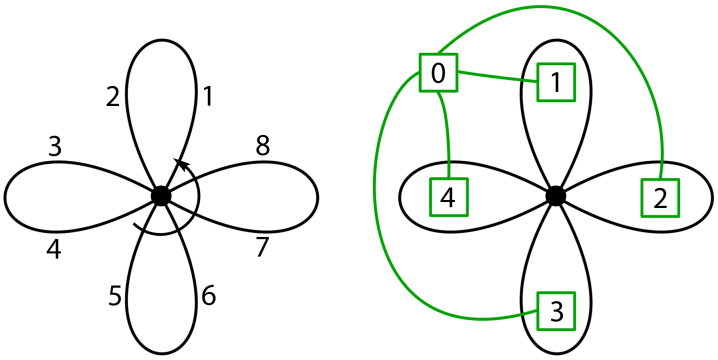
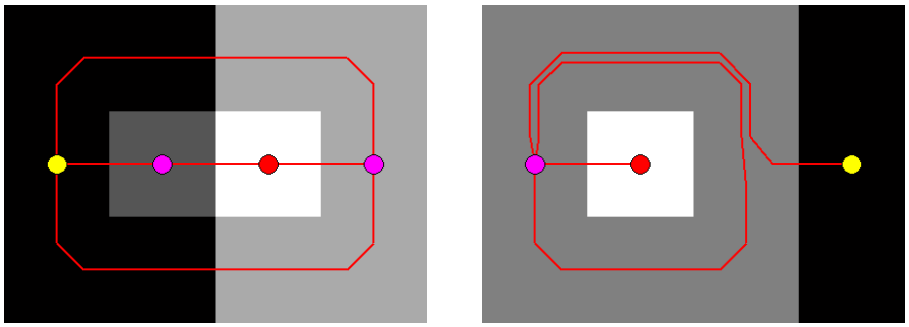


Figure 2.8: An example showing the topology preserving property of a combinatorial map. The left image shows an object encoded as a combinatorial map with $\sigma = (1, 2, 3, 4, 5, 6, 7, 8)$, and the right image shows an encoding using dual graphs. Here the primal graph is drawn in black its dual in green. By switching any of the loops of the object the σ-permutation would change leading to a new combinatorial map, but the new dual graph would be isomorphic to the unaltered one.

The reordering of the darts takes comparably longer than applying a reduction operation, since the operation only changes maximally 2 values in the σ-permutation, and the reordering of the darts has to relabel in the worst case all of the darts. The only two cases where no reordering has to be performed is when the "last" edge in the graph is being reduced, or the combinatorial pyramid has been already once reduced to the apex and reconstructed. At this point it is only necessary to apply the reduction operations. Out of this reason we can choose an algorithm for constructing contraction kernels with a high reduction factor, for example [HGS+02].

2.3 Topology Preservation

As it could be seen in the previous sections, the combinatorial pyramid is a more concise representation of the dual graph pyramid. The combinatorial pyramid inherits also one of the most important properties from the structural point of view, namely topology preservation [BK99, BK03b]. To visualize this property, an example object is encoded using a combinatorial map and dual graphs in Figure 2.8. The advantage of the combinatorial map over dual graphs comes from its embedding in the plane. Imagine that any two loops of the object would be exchanged. In the combinatorial map encoding this would lead to a different σ-permutation and thus a different object. In the dual graph representation however this exchange would produce a new dual graph which is isomorphic to the original one, meaning that the new object would be structurally equal to the unaltered one. This way the topological representation of an object using a combinatorial map is unique. In a combinatorial pyramid each of the levels consists of a combinatorial map which encodes the current partitioning of the image. When the pyramid is additionally encoded using the canonical representation, the history of reduction operations is captured as well. This



(a) (b)

Figure 2.9: A visualization of (a) multiple borders and the (b) inclusion relationship captured by the combinatorial pyramid. Notice in the top image multiple borders between the two largest regions, as well as the self-loop of the large region around the small included region in the lower image. The coloring of the vertices indicates the topological class of each vertex, discussed in more detail in Section 3.2. A yellow vertex color indicates a local minimum, red a local maximum, and pink a singular slope.

way an unique relationship between individual levels of the pyramid is produced without loss of functionality [TK14]. This ultimately leads to the ability of the combinatorial pyramid to fully reconstruct the initial combinatorial map from any level of the pyramid.

Additionally the pyramid differentiates between adjacency- and inclusion relationships of regions, and between single- and multiple borders between regions. To visualize these properties (and combinatorial pyramids in general) we implemented the algorithm proposed in [IIKG08]. A simple example of the ability to capture multiple borders between regions can be seen in Figure 2.9a, and an example of the inclusion relationship can be seen in Figure 2.9b.

2.4 Image Graphs

As it is said in [BK03b], the efficiency of an irregular pyramid is strongly influenced by two closely related features:

1. The decimation process used to build one graph from the graph below, and
2. The data structure used to encode each graph of the pyramid.

The decimation process used during the construction of the pyramid defines on one hand the height of the pyramid, and on the other hand the properties, that may arise from the decimation process. As was shown in [HGS+02], the height of the pyramid can be bounded by logarithmic height, which brings the reduction factor of the irregular pyramid on the level of the regular pyramid. Other properties, that may arise from the

decimation process are for example parallelization, uniform contraction kernel size, and others. The data structure that is used to encode the graphs at each level of the pyramid determines the properties that may be encoded with each of these graphs. Simple region adjacency graphs and dual graphs do not have the ability to encode multiple borders and the inclusion relationship. Combinatorial maps encode the graph as a set of darts, explicitly encode the dart neighborhood orientation, and implicitly encode the dual combinatorial map. Generalized maps are able to represent non-orientable surfaces and open subdivisions. The combination of the chosen decimation process and the data structure to reduce and encode the graphs at each level of the pyramid define the overall properties of the irregular pyramid.

In this work we define a new decimation process which satisfies the needs of a structurally correct image segmentation, defined later in Chapter 5. The underlying data structure to store the graphs of the pyramid will be combinatorial maps, with an additional canonical representation for compressing the storage required for capturing the reduction history. Combinatorial maps have the advantage, that they implicitly encode the dual combinatorial map. This way the dual combinatorial map may be retrieved at any time from the primal combinatorial map, and vice-versa. Another property of the combinatorial maps is, that they are equivalent to dual graphs [BK99], and are at any time interchangeable with these dual graphs. *For this reason, many of the concepts in this work will be explained in terms of dual graphs, even though the main underlying data structure will be a combinatorial map.*

A digital image can be very easily translated to a (primal) image graph, as illustrated in Figure 2.10. Each pixel in the image is represented in the graph $G = (V, E)$ as a vertex $v \in V$, and the neighborhood relationships are represented as a set of edges E. Assuming 4-neighborhood, the corner pixels have 2 neighbors, the remaining border pixels have 3 neighbors, and the internal pixels have 4 neighbors. This way it is possible to draw the image graph G in one plane with the mentioned neighborhood relations. Since for the presented image segmentation algorithm in this work it is crucial to have encoded multiple edges and inclusion relationships, it is necessary to encode the image using a dual graph or a combinatorial map. For this reason, the dual image graph $G' = (V', E')$ is constructed, where the vertices V' correspond to 2×2 pixel configurations called faces, and the edges E' connect these pixel configurations. At a higher level k of the irregular pyramid, the vertices V_k correspond to whole regions, i.e. connected components of pixels. These regions may vary in shape and size, and at the semantic level they may represent a segmented object in the image. Equivalently, this dual concept can be translated to combinatorial maps. Each vertex $v \in V$ of graph G is encoded in the combinatorial map $C = (\mathcal{D}, \alpha, \sigma)$ by a σ-orbit $\sigma^*(d)$, where d is a dart associated with vertex v. The edges E of graph G are encoded as dart-pairs, each connected by an α-orbit $\alpha^*(d)$. The advantage of the combinatorial map is, that the dual map $C' = (\mathcal{D}, \alpha, \varphi)$ can be easily computed. The dart set $\mathcal{D}$ and the α-permutation remain the same, and the φ-permutation may be computed as $\varphi = \alpha \circ \sigma$ or $\varphi = \sigma \circ \alpha$.

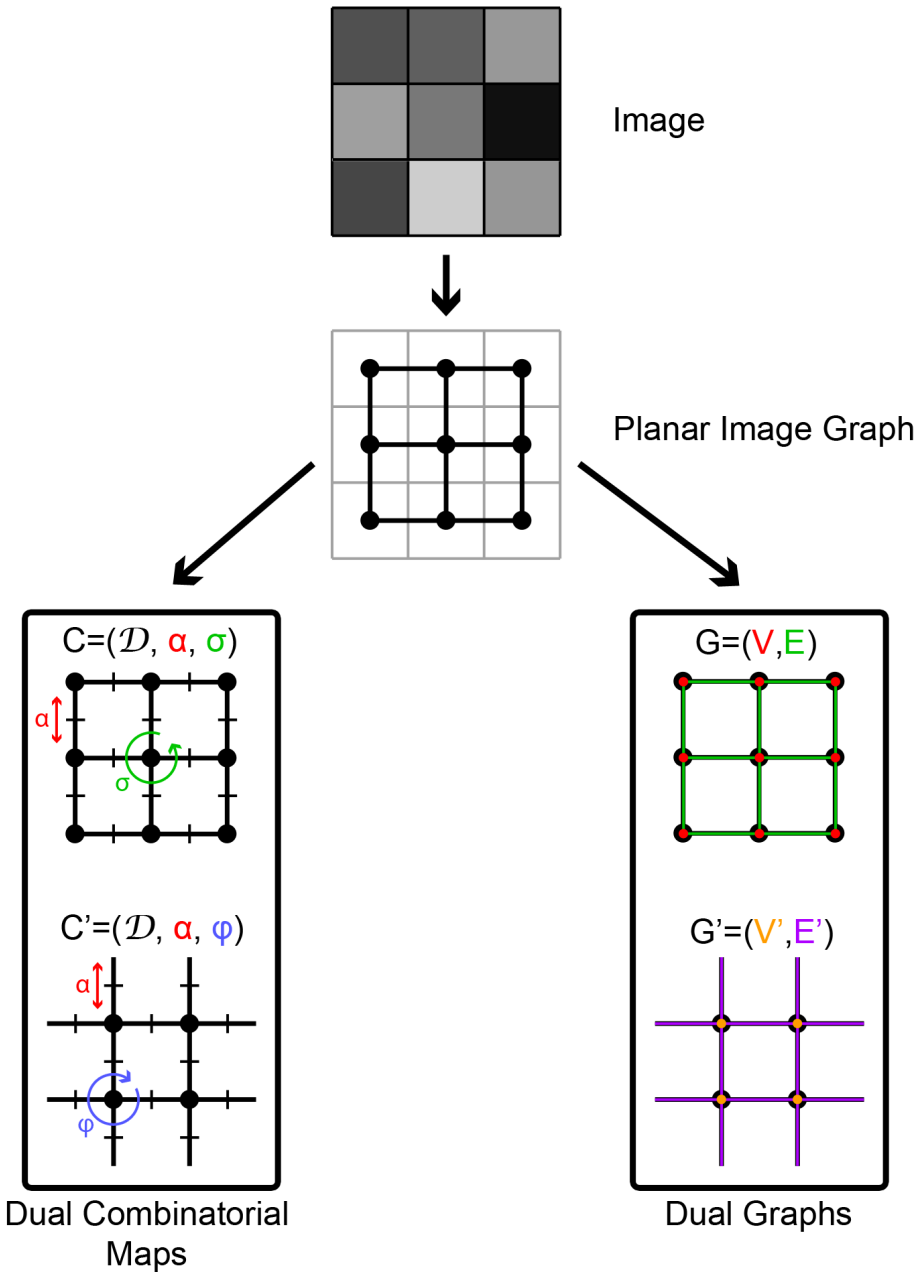


Figure 2.10: An illustration showing the dual graph and dual combinatorial map representation of an image. The vertices of the image graph encode the pixels of the image, and the edges the neighborhood relationships. The vertices of the dual graph, which can be constructed from the primal image graph, encode 2×2 pixel configurations. The dual graph and dual combinatorial map representations of the image are equivalent [BK99] and thus interchangeable.

CHAPTER 3

Local Binary Patterns (LBP)

Texture is omnipresent in the real world and for humans an intuitively understood concept, yet in the technical world there is still a lack of a generally accepted definition of texture. In the scientific literature there are many different attempts to define texture, however these are still relatively vague, for example:

- "Texture refers to properties that represent the surface or structure of an object.(...)We might define texture as *something consisting of mutually related elements*." [SHB93],
- "A texture area in an image can be characterized by a non-uniform or varying spatial distribution of intensity or color." [PZHA11],
- "Texture is an important property of images, representing the structural and statistical distribution of elements throughout the image." [Wal14].

In general there are two types of textures, namely regular- and irregular textures. Regular textures consist of ordered repeating structural elements, such as a brick wall, and irregular textures follow a statistical distribution, but do not have any repeating pattern [Wal14], for example pebbles on a road. Textures also usually consist of multiple levels of structure. A good example for this is a checkered piece of cloth, where the large scale structure is the pattern made of the textile checks, and the small scale structure being individual stitches from which the textile is woven. Another example is an irregular stone wall, where the large scale structure defines the arrangement of the rocks, and the small scale structure describes the surface of individual rocks. Images may contain a single texture, for example a brick wall, or multiple textures, such as a satellite image of a terrain with urban- and forest areas. The problem of automatically segmenting and classifying images with texture dates back at least 40 years [WDR76]. Already at that time, texture classification algorithms could be divided into structural and statistical approaches. Structural approaches computed features derived from the texture's

Fourier power spectrum, whereas statistical approaches computed features from a gray level co-occurrence matrix (GLCM) known as Haralick features [HSD73] or performed a classification based on local properties measured on the texture such as the mean and the standard deviation of pixels. Later in 1990, He and Wang [HW90] introduced the notion of a Texture Unit, which describes the local relationship of a pixel to its 8 neighbors. Texture Units distinguish $6561 = 3^8$ individual local texture elements, which when statistically described using a histogram could classify a texture. This classification method inspired Ojala et al. [OPH96] in 1996 to introduce the Local Binary Patterns (LBP), which quickly evolved into one of the most studied and modified texture descriptors. The functionality of the Local Binary Patterns is the same as of the Texture Units with the difference, that less ($256 = 2^8$) local texture elements can be distinguished. Since the introduction of the Local Binary Patterns numerous different extensions of this texture descriptor have been proposed, e.g. to guarantee rotation invariance [OPM02], enable multi-scale analysis [TP06], or gain the ability to process color images [MP04]. Further research also included the combination of the LBP with other local feature descriptors like for example SIFT [Low99, HPS06] to increase the matching accuracy between objects in different images. With this large base of modifications and extensions of the LBP descriptor it is possible to apply it to many applications, e.g. for face recognition and verification, medical image analysis, background subtraction, human detection, gender classification, and facial expression recognition [PHZA11].

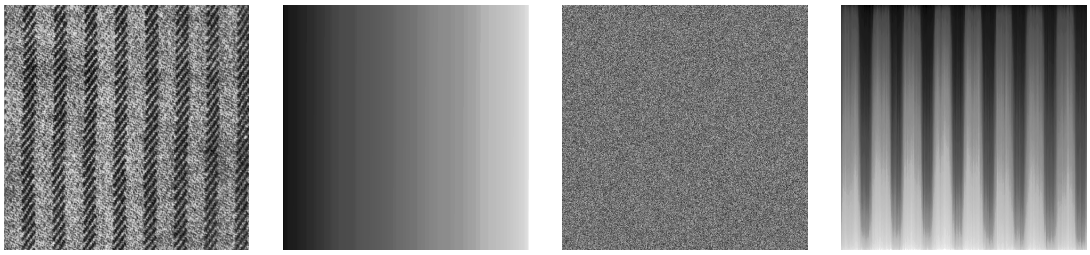
This chapter is organized as follows: Section 3.1 describes the original Local Binary Pattern texture descriptor and explains the constraints the LBP codification of an image sets on the histogram of this image. Section 3.2 describes the classification of primal and dual vertices into five topological classes based on their associated LBP code. Section 3.3 explains how it is possible to transition to strict inequality Local Binary Patterns using the combinatorial pyramid framework, and finally Section 3.4 handles the special case of non-empty self-loops for the assignment of one of the five topological classes.

3.1 Local Binary Patterns (LBP)

The original Local Binary Pattern texture descriptor was proposed in 1996 by Ojala et al. [OPH96] as a way to locally describe the texture around each pixel in an image. The idea is, that each distinct local texture element can be described as a sequence of 0- or 1-bits, which can be interpreted as a decimal number. Let the intensity of a pixel $p = (x, y)$, denoted by $g(p)$, be encoded using 8 bits, i.e. as an integer value in the range $[0, 255]$. Then the original decimal LBP is computed for a (center) pixel as:

$$LBP_P = \sum_{i=0}^{P-1} s(g(p_i) - g(c))2^i \,, \tag{3.1}$$

where P is the number of neighbors, c is the center pixel of the operator and p_i is the local neighbor indexed by i. The basic operator uses the sign function $s(x)$ with:



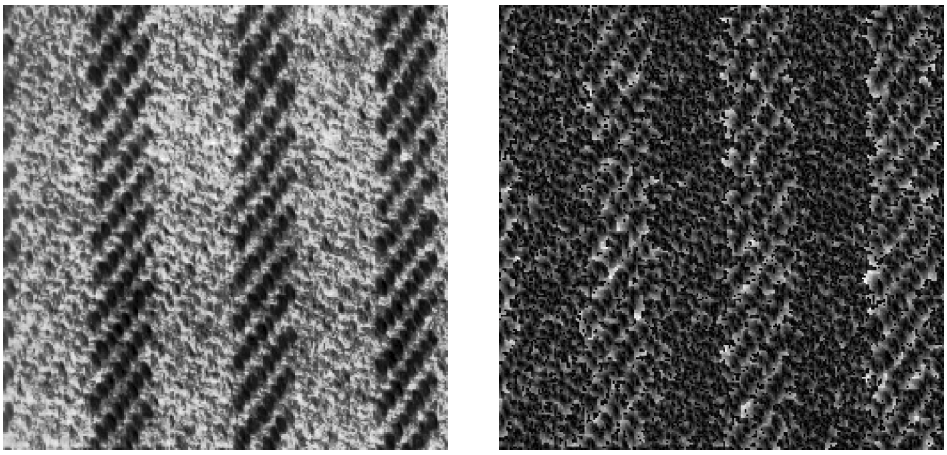
(a) (b) (c) (d)

Figure 3.1: An example of visually different, but statistically equivalent images. All of these images have the same gray-value histogram. (a) is the original image, (b) are the ordered pixels by values, (c) has a random permutation of the location of the pixels, and (d) has ordered pixels by values by columns.

$$s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \tag{3.2}$$

The authors of the original LBP proposed to use the local 3×3 neighborhood for the computation, i.e. $P = 8$. The Equation 3.1 can be interpreted as a sequence of three operations, namely first the 8 neighbors of a pixel are thresholded by its value, then the resulting sequence of 0- and 1-bits is multiplied by increasing powers of 2, and finally the resulting weighted numbers are summed up to represent the local texture around the center pixel as a unique decimal number. This way, in the case of a 3×3 neighborhood $2^8 = 256$ distinct local texture elements can be identified and described.

The standard approach to represent a region using the LBP is to describe it statistically using a histogram. In case of the presented LBP this is a histogram with 256 bins. To perform a classification of this textured region, a comparison with a set of known samples using a histogram distance measure is performed. When the set of known textures is structurally diverse with respect to the LBP codification of the image, this approach performs well. Problems arise, when two textures have a similar amount of the same LBP codes scattered around at different spatial locations. These textures may seem to us humans to be different and clearly distinguishable, but from a statistical point of view these textures may seem very similar. This is the main problem of the histogram, namely that it is not a bijective function and it does not take the spatial locations of its values into consideration. An example of this principle can be seen in Figure 3.1. By encoding the image using the Local Binary Patterns, the local structure of the image is taken into consideration. Every single permutation of values in the context of smaller/greater and equal than the center pixel is codified using a unique number. This puts considerable restrictions on the size of the class of images which have the same LBP histogram. Also, the spatial location of the locally captured structure by the LBP is lost when using a global histogram. An example of images with equal LBP histograms



(a) (b)

Figure 3.2: An example of images with the same LBP histogram. (a) is the original image, (b) is the original image after applying a LBP preserving transformation.

can be seen in Figure 3.2. Observe, that the images are visually different when comparing the grayscale values, however the same basic structural elements like edges and corners are present in both images.

3.2 LBP Classes

As mentioned in the previous section, after thresholding the peripheral values by the center value, the binary numbers are brought into sequence to form an 8-bit binary number. The length of the binary LBP number depends on the number of neighbors of a pixel when using the original LBP, or on the sampling rate of a more generalized version of the LBP [OPM02]. This extended version samples the local neighborhood of a pixel at a specified radius and sampling rate, enabling multi-resolution analysis and rotation invariance of the texture descriptor. The original 4-neighborhood LBP texture descriptor can be computed with the extended LBP texture descriptor by using the radius one and a sampling at 4 points. The problem with the generalized version is that it is only defined on a square grid, which corresponds in the combinatorial pyramid representation to the base level. At a higher level of the pyramid, to compute the more generalized LBP of a region, one would need to increase the number of sampling points to achieve the same sampling rate at the border of the region, which would automatically lead to a variable length of the binary LBP code. An additional problem might occur at concave regions, where some of the sampling points would be inside the same region. Out of this reason the original LBP will be used in this work.

When an image is represented as a graph, the length of the original LBP number of a vertex depends on the number of edges incident to that vertex. In case of the

combinatorial pyramid representation this translates to the cardinality of the σ-orbit of each encoded vertex. At the base level of the pyramid the corner vertices have 2 neighbors, the remaining border vertices have 3 neighbors, and the rest of the vertices have 4 neighbors. At the level k of the pyramid with simplification, this number of vertices can however range from 1 to n neighbors. In case of the LBP code of an encoded vertex corresponding to a region in the image, the LBP code would be represented as a 1- to n-bit binary number.

All of the LBP codes can however be classified into 5 topological classes regardless of the number or circular shifting of the bits of a LBP code. To give an example, imagine a gray-scale image with a Gauss-blurred dot. When this image is interpreted as a height-map, then the original position of the dot corresponds to the peak of a mountain, which corresponds to a local maximum. When this region is merged with regions in a circular neighborhood, then the new region will look like a flattened peak, but will still correspond to a local maximum. The length of the LBP code of this new region will be longer, but the classification remains the same. This classification also remains the same, when the binary numbers are padded with the first- or last binary value, so that all of the LBP codes have the same length for comparison or other reasons.

For the classification of the LBP codes we will be assuming *strict inequality* relationships, that means a 0-bit expresses the smaller than ($<$) relationship of a center vertex to the peripheral vertex, and a 1-bit the greater than ($>$) relationship. This assumption is made to avoid problems caused by the asymmetry of the 0- and 1-bits of the LBP code. These problems and a way to solve them will be discussed in the next section. The 5 classes shown on an example of 4-bit LBP codes are:

- *Minimum*: [0000],
- *Singular Slope*: [1000],[0100],[0010],[0001],[0111],[1011],[1101],[1110],
- *Slope*: [1100],[0110],[0011],[1001],
- *Saddle*: [1010],[0101], and
- *Maximum*: [1111].

The notation "[.....]" represents the sequence of 0- and 1-bits stored at the individual darts of an orbit. For example [0011] means, that when traversing the orbit in a specified direction (CW, CCW), then the center pixel is smaller than 2 of its adjacent neighbors and larger than 2 of its remaining adjacent neighbors.

In general, the class of the LBP code can be assigned by counting the number of 0/1 and 1/0 transitions. Minima and maxima are defined by zero transitions, slopes by two transitions, and saddles by four or more transitions. Slopes can further be divided into a regular- and a singular slope. Singular slopes are slopes, where there is only a

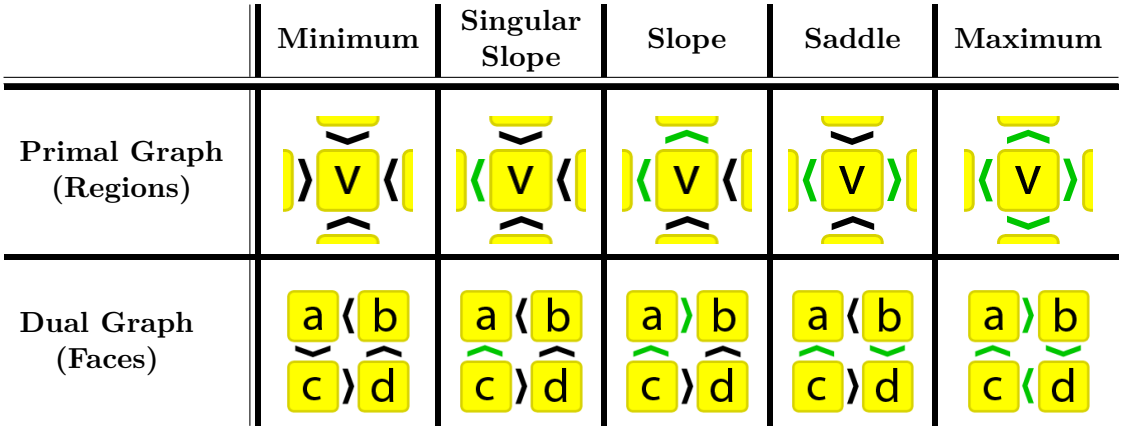


Figure 3.3: An example of the classification of vertices in the primal and dual graph into five topological classes. The top row shows vertices of the primal graph, that correspond to regions, and the bottom row shows vertices of the dual graph, which correspond to faces in the primal graph. The degree of the dual vertices is in all cases 4, which means that each face is bound by 4 edges. The arrows between the vertices display the strict inequality relationships between the vertices. 0-bits are marked black, and 1-bits are marked green. Notice, that in any correctly constructed image graph, it is not possible that dual local extrema occur (for more details see Section 4.4).

single occurrence of a 0- or 1-bit. The remaining slopes are classified as regular slopes. Ojala et al. [OPM02] refer to our defined slopes as "uniform" Local Binary Patterns, and based on their findings this LBP class represents the vast majority of classes found in images with texture, sometimes even over 90 percent. An analysis of the topological classes in randomly selected natural images can be seen in Section 5.4. The top row in Figure 3.3 illustrates the classification of a primal vertex corresponding to a region in the image into each of the topological classes based on its binary neighborhood relationships.

In the combinatorial pyramid representation, the relationships between two adjacent vertices can be implemented as binary attributes of the corresponding darts $l(d)$. Here d is a dart at the center vertex, for which $l(d) = 0$ if there is a smaller ($<$) than relationship between the center vertex and its α-connected neighbor, and $l(d) = 1$ if there is a greater or equal than ($\geq$) relationship. In combinatorial pyramids with only strict inequalities, $l(d) = 1$ describes a greater than ($>$) relationship.

The topological classification of LBP codes can be also applied to the vertices of the dual graph which correspond to faces in the primal graph, and thus classify each of these faces. The incident edges of a dual vertex represent the bounding edges of a face, and the binary relationships stored at these edges define the binary relationships between the vertices incident to the bounding edges of a face. In the combinatorial map representation, the bounding edges of a face are split into pairs of darts. Each of these darts is associated

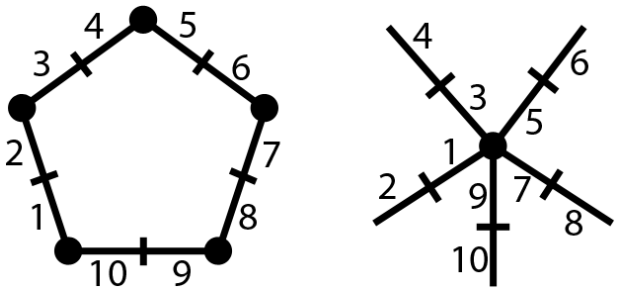


Figure 3.4: A simple graph encoded as a combinatorial map with its dual map on the right with $\varphi = \alpha \circ \sigma$. In this example the single inner face of the graph is bounded by five edges, which are split up into 10 darts. As can be seen on the right dual map, the face is defined by the odd numbered darts.

with exactly one face, and thus each dart pair identifies the two faces incident to the edge formed by this dart pair. All of the darts associated with a single face are the defining darts of this face, and they are included in the dual orbit $\varphi^*(d)$.

Definition 9. Face extrema

Let a face in the primal graph be encoded by $\varphi^*(d)$ in the dual combinatorial map $\overline{C} = (\mathcal{D}, \alpha, \varphi)$. Let there be only strict inequality relationships stored at each dart d as binary attributes $l(d)$. Then the vertex associated with dart d is a face extremum if:

$$\textbf{\textit{Face Minimum:}} \begin{cases} l(d) = 0 \ \wedge \ l(\varphi^{-1}(d)) = 1 & \text{if } \varphi = \alpha \circ \sigma \\ l(d) = 0 \ \wedge \ l(\varphi(d)) = 1 & \text{if } \varphi = \sigma \circ \alpha \end{cases}$$

$$\textbf{\textit{Face Maximum:}} \begin{cases} l(d) = 1 \ \wedge \ l(\varphi^{-1}(d)) = 0 & \text{if } \varphi = \alpha \circ \sigma \\ l(d) = 1 \ \wedge \ l(\varphi(d)) = 0 & \text{if } \varphi = \sigma \circ \alpha \end{cases} \tag{3.3}$$

To get a better understanding of the face extrema, observe the illustration in Figure 3.4. In this example, the left image shows a simple graph encoded using a combinatorial map, and on the right side its dual map with $\varphi = \alpha \circ \sigma$. The inner face of the primal graph is bounded by five edges, which are split up into ten darts. Assuming only strict inequality relationships, then either $l(d) = 1 \ \wedge \ l(\alpha(d)) = 0$ or $l(d) = 0 \ \wedge \ l(\alpha(d)) = 1$ has to apply. Let v be the primal vertex associated with dart number 1. Then v is a face minimum, if $l(1) = 0$ and also $l(10) = 0$. In the dual combinatorial pyramid the darts 1 and 10 are connected by the operation $\alpha(\varphi^{-1}(1)) = 10$. From this follows, that a face minimum is defined by $l(1) = 0$ and also $l(\alpha(\varphi^{-1}(1))) = 0$. Since only strict inequality relationships apply, $l(10) = 0$ can be replaced by $l(9) = 1$. This operation removes the α-operation from $l(\alpha(\varphi^{-1}(1))) = 0$, which means that the vertex v is a face minimum if $l(1) = 0$ and also $l(\varphi^{-1}(1)) = 1$, as defined in Equation 3.3. The process for defining a face maximum, as well as the face extrema with $\varphi = \sigma \circ \alpha$ can be derived analogously.

Same as with the primal LBP classes, dual topological minima and maxima have zero 0/1 or 1/0 transitions, slopes are defined by one 0/1 and one 1/0 transition, and saddles are defined by more than two of each transition. The bottom row in Figure 3.3 illustrates the classification of a face into each of the topological classes based on the binary relationships of its bounding edges. Observe in this figure in the bottom row the face, that is a slope. In this face the vertex b is a face minimum and the vertex c is a face maximum.

3.3 Strict Inequality LBP

The problem with not using strict inequalities is that ambiguities in the classification of a LBP code may arise because of the 1-bit. An example is with the primal topological maximum, where the LBP code consists of a sequence of 1-bits. At this point it is not clear whether the vertex is larger than all surrounding vertices, equal to all of them, or some of them are smaller and some are equal to the center vertex. To clarify if the center vertex has an equal value to a neighboring vertex, it is necessary to check if $l(d) = l(\alpha(d)) = 1$, where d is a dart at the center vertex and $\alpha(d)$ is its α-related counter-dart at the neighboring vertex.

The solution to these problems is to merge neighboring regions with equal values called *plateaus*.

Definition 10. Plateau

*Let $\alpha^*(d)$ correspond to an edge in the primal graph encoded by a combinatorial map $C = (\mathcal{D}, \alpha, \sigma)$. Let u and v be the associated vertices with the darts d and $\alpha(d)$, and let $l(d)$ and $l(\alpha(d))$ be the binary attributes storing the non-strict relationship between u and v. Then u and v form a **plateau**, if $l(d) = l(\alpha(d)) = 1$.*

Due to the asymmetry of the 0- and 1-bits the case $l(d) = l(\alpha(d)) = 0$ cannot occur and thus does not need to be checked. *From the structural point of view the merging of plateaus has the consequence, that all of the relationships between vertices change to strict inequalities.* The only side-effect is, that the binary attributes $l(d)$ stored at the darts of the self-loops misrepresent the true relationships between the incident vertices. The problem is, that if u is the vertex associated with dart d and vertex v with $\alpha(d)$, then the attributes $l(d) = l(\alpha(d)) = 1$ would mean $u > v \wedge v > u$. Self-loops are however special edges and will be discussed more in detail in the next section. *For the remainder of this work we will be assuming strict inequality relationships between regions, that can be achieved by merging plateaus.*

3.4 Self-loops in Strict Inequality LBP

After merging plateaus and simplifying the combinatorial pyramid using the dual reduction scheme, only non-empty self-loops remain. Non-empty self-loops are special edges, because

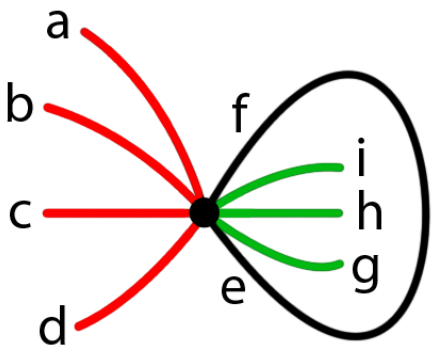


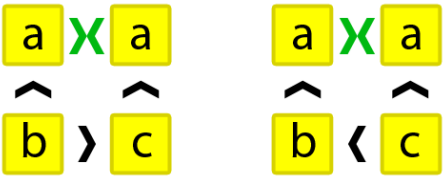
Figure 3.5: An example division of darts around a vertex with a non-empty self-loop into inner and outer darts. Darts marked with red color are outer darts, and darts marked with green color are inner darts.

they describe the inclusion relationship, a contraction of these edges is not possible without destroying the connectivity of the primal or dual graph [BK99], they consist of the only darts which have an equality relationship, and they separate darts at a vertex with a non-empty self-loop into *inner* and *outer* darts. The last property allows to assign a vertex with a non-empty self-loop LBP classes in a local- and global context. The LBP class can be assigned by considering:

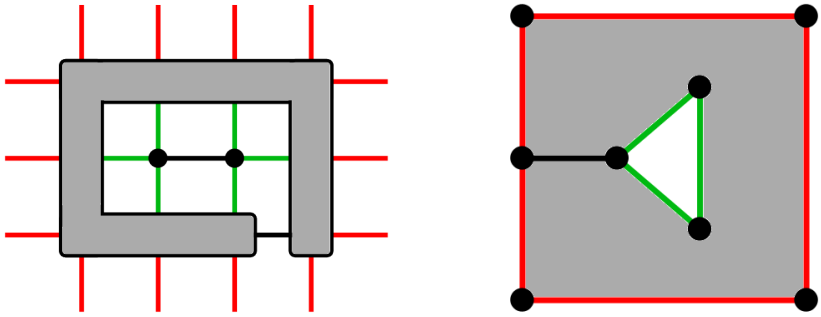
- Only the inner darts,
- Only the outer darts, or
- All darts except for the self-loop.

Figure 3.5 shows a subdivision of darts around a vertex with a non-empty self-loop into inner and outer darts.

During the assignment of the LBP class the self-loop has to be ignored, because otherwise ambiguities may arise. This is due to the fact, that the inequality relationships $l(d)$ of the darts of the self-loop are no more valid. Consider the following case, where after the contraction of plateaus the remaining non-empty self-loops have the binary relationships $l(d) = l(\alpha(d)) = 1$:



In both faces the top vertices are part of a large region, and the incident edge is a self-loop. The arrows signify the direction of the edges based on the 0- and 1-bits stored at the



(a) (b)

Figure 3.6: An illustration of the subdivision of inner and outer darts in the (a) primal graph, and the (b) dual graph. Vertices containing the non-empty self-loop are filled with gray color, darts marked with red color are outer darts, and darts marked with green color are inner darts.

darts. The arrows of the top edge are highlighted in green color, as both of them store 1-bits. When traversing the dual dart orbit in clockwise direction, the left face has the LBP code [0011] and the right face has [0111]. This way the left face is a slope, and the right face a singular slope. When traversing the faces in counter-clockwise direction, the left face has the LBP code [0111] and the right face has [0011]. In this case the LBP classes of the two faces are switched. To prevent such ambiguities, the self-loops have to be ignored. In the above example, when ignoring the self-loop, both LBP codes have the length 3 and become singular slopes.

In the primal graph a vertex with a non-empty self-loop corresponds to a merged region which encloses one or more regions. In the global context when considering all surrounding darts of the vertex, both the outer vertices and the enclosed vertices are taken into consideration for deciding the LBP class of the region. In the local context when considering only the outer darts, the merged region and all its contents are perceived from outside as a single vertex. Analogously, when considering only the inner darts, the merged region is perceived from inside as a single enclosing region. In the dual graph a vertex with a non-empty self-loop corresponds to a face, which has an outer border, touches at the edge corresponding to the self-loop, and has an inner border. In the local context, only the inner or outer border are perceived. Figure 3.6 illustrates a vertex with a non-empty self-loop in the primal as well as the dual graph, and highlights the inner and outer darts.

The decision whether to subdivide the darts depends on the context of the application. In this work we will use this subdivision to assign a LBP class to a vertex based on whether the currently observed dart is an inner or an outer dart.

CHAPTER 4

Image Structure

So far the theory of combinatorial pyramids and Local Binary Patterns has been explained. Using these two concepts it is possible to define a framework for the task of image segmentation, which utilizes the structure of the image as a basis. In the following chapter we will define the structure of an image using LBPs in Section 4.1, which allows to make decisions whether two images belong to a set of images with the same structure. Since this set can become large, we also describe an algorithm to compute a representative image for each such set. Section 4.2 describes two types of redundant structural information and a way to remove dual singular slopes. Section 4.3 shows how it is possible to remove dual saddles. Using the theory in these two sections it will be possible to generate an image representation with only dual slopes. Finally, Section 4.4 describes the theory of the previous sections using concepts from the order- and graph theory, and defines a structurally correct image segmentation.

4.1 Structural Equality and Representative Image

In this work we define the structure of an image at the lowest possible level, namely as binary relationships between individual regions. Our reasoning is, that we first try to segment the micro-structures of the objects in the image, and then successively merge these micro-structures to identify structures at a higher level. Additionally by not introducing structural errors, which will be defined in the next section, we expect to get a segmentation with plausible hierarchical relationships between individual levels of structure in the image. An analogous example in the real world is that when segmenting a tree, we first try to identify the individual leaves and bark of a tree, which when unified form the stem and crown of the tree. The LBP enable us to define the structure of an image on such a low level.

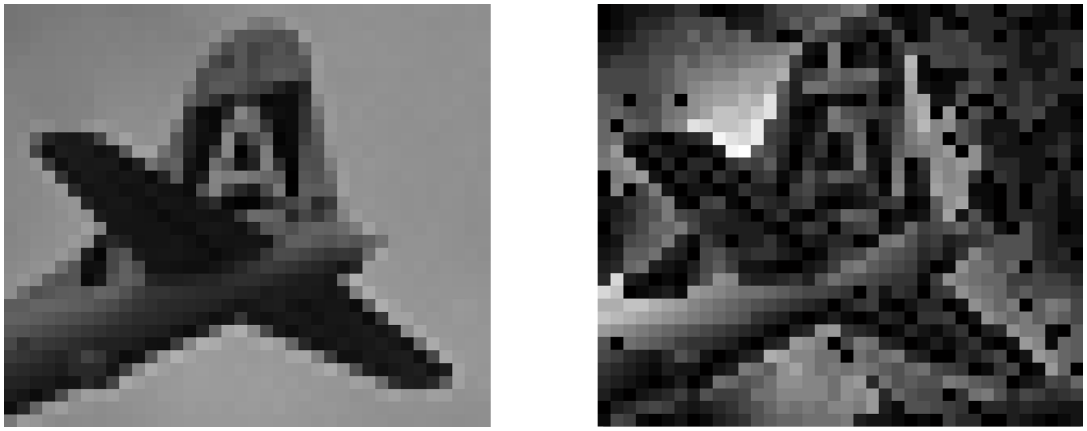


Figure 4.1: An example of two images which have the same structure and thus the same LBP codification. Notice that even though these images are visually different, structural elements such as edges or corners are preserved.

Using the original Local Binary Patterns [OPH96], an image I can be *LBP codified* to an 8-bit grayscale image $LBP(I)$ to represent local texture elements at each pixel in I. By considering only the structure of the image using the LBP codification, there is a whole class of images S which have exactly the same LBP codification, and are thus *structurally equal*:

$$S = \{(I, J) \mid LBP(I) = LBP(J)\}, \tag{4.1}$$

where I and J are two images of the same dimensions. Figure 4.1 shows an example of such two images which have the same structure and thus the same LBP codification, but are visually different.

The class of images S with the same structure as the input image I is usually very large, and therefore we define a *representative image* $REP(I) \in S$ for each class:

$$REP(I) = REP(J), \quad I \neq J, \quad I, J \in S, \tag{4.2}$$

from which also follows

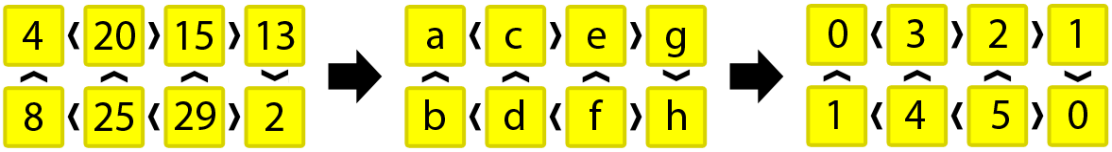
$$LBP(REP(I)) = LBP(I). \tag{4.3}$$

The algorithm for the computation of the representative image is a general algorithm for computing the longest paths in directed graphs without cycles. These graphs can be constructed by first representing the original image as an undirected image graph,

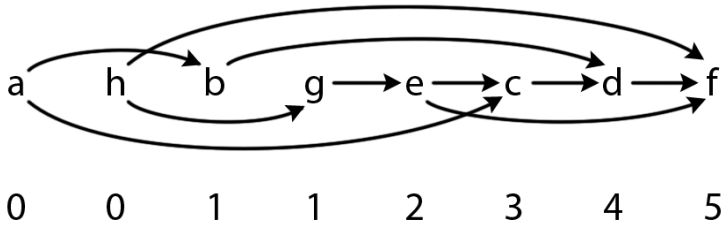
Algorithm 4.1: Algorithm for computing the representative image REP

Data: Directed Acyclic Graph $G = (V, E)$
Result: Representative image REP encoded as attributes $r(V)$

```
1 Initialize vectors TS, I;
2 Initialize all attributes r(V) := 0;
3 TS := TopologicalSort(G);
4 forall the vertices v ∈ TS do
5 |   I := findIncomingEdges(v);
6 |   forall the edges (s, v) ∈ I do
7 |   |   r(v) := max{r(v), r(s) + 1};
8 |   end
9 end
```



(a)



(b)

Figure 4.2: An example of the computation of the representative image using Algorithm 4.1. (a) shows the original image on the left and the final representative image on the right, and (b) shows the topologically sorted vertices and their representative values on the bottom.

and after merging vertices with equal values, each edge in the image graph is assigned a direction. As it is shown in more detail in Section 4.4, the resulting graphs contain no cycles. The pseudocode for this algorithm can be seen in Algorithm 4.1 and an illustration of its functionality in Figure 4.2. This algorithm can be applied to a combinatorial map with just slight modifications, since the combinatorial map encodes the edges of a graph as darts, and the vertices as orbits of darts.

In this algorithm, the vertices of the graph G are first topologically sorted in line 2. Since G is a DAG, at least one topological ordering must exist. The topological sorting is

a well-known problem in graph theory and there exist a number of algorithms, among others by means of the Depth-First Search algorithm, which provide a solution that is linear in time with respect to the size of the graph, i.e. $\mathcal{O}(|V|+|E|)$. The for-loop in lines $3-12$ then traverses each vertex once and looks if it has any incoming edges. If there are no incoming edges, then this vertex is a local minimum, and receives the value 0 in the representative image. Otherwise, the current vertex receives the next highest value from all of its predecessors. This for-loop processes each vertex and edge once and is also linear in the size of the graph, $\mathcal{O}(|V|+|E|)$. Note, that this algorithm requires a DAG as input. Since a combinatorial map after merging plateaus may include self-loops, that are only present to preserve the topology of the image and are ignored during the computation of the LBP class of a vertex, these edges need to be removed prior to computing the representative image. This pre-processing step has the complexity $\mathcal{O}(|E|)$, since all edges need to be checked if $(v, v) \in E,\ v \in V$ is true.

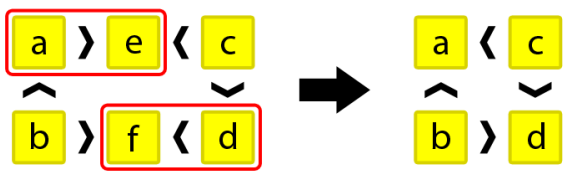
4.2 Structural Redundancy

As was mentioned in Section 3.2, each 0/1 and 1/0 transition in the dual LBP code defines a face minimum or face maximum in the primal graph. All structural information inside a face is stored as directed paths between such minima and maxima. Because dual topological minima and maxima describe directed cycles, which cannot naturally occur in images, the only allowed LBP classes of faces are singular slopes, regular slopes, and saddles. This way all naturally occurring faces have at least one 0/1 and 1/0 transition.

There are two types of operations which can be applied with respect to a face, that do not change the structural information content:

1. Contraction of an edge which does not connect a face minimum and a face maximum, and
2. Removal of an edge corresponding to a single occurring 0- or 1-bit in a dual singular slope.

The first operation exploits the transitive property of the LBP. The inequality relationship between two regions connected by a directed path is independent of the length of these paths. This way no structural information is lost, when these paths get reduced to the length 1:

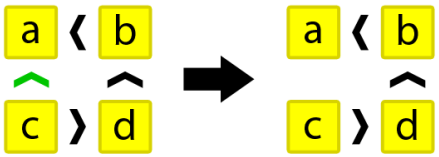


Proposition 1 *Let there be a directed path of length ≥ 2 between a face minimum and a face maximum. By contracting any two adjacent vertices along this path, the LBP class of the face containing this path may change only if it was a slope, otherwise the LBP class remains the same.*

Proof of Prop. 1 *The length of a directed path along the border of a face is given by the length of consecutive 0- or 1-bit sequences in the orbit of the dual vertex corresponding to the face. Paths of length 1 correspond to single occurring 0- or 1-bits. By contracting only paths of length >= 2, all of the 0/1 and 1/0 transitions in the dual orbit remain preserved, and thus the LBP class of the dual vertex does not change if it was a saddle. Singular slopes may only be reduced at the longer path, thus preserving the characteristic single occurring 0- or 1-bit and also its LBP class. In case of a slope, both paths may be reduced to a length 1, resulting in the LBP code* [01]*, which is a singular slope.* ■

In this work we will only be using the second reduction operation, because we reserve the contraction in the primal graph purely for the purpose of expressing the similarity of two regions, and thus their belonging to the same region describing an object in the image. *For this reason, when we will be talking about the "removal of a redundant edge", we will be always referring to the second reduction operation.*

This second operation is based on the fact, that every face that is a slope, has two paths between its face minimum and face maximum. Singular slopes have additionally the property, that one of these paths has the length 1. By removing this shorter path, all of the inequality relationships between the vertices of a face remain preserved, because there exists a second path from the face minimum to the face maximum:



When looking at a face minimum, then it is known that the LBP code of this vertex has to contain at least two consecutive 0-bits, [...00...]. Based on the degree of the orbit of this vertex, the LBP class of this vertex may vary. In case the degree of the orbit of this vertex is 2, then this vertex can only be a local minimum [00]. If the degree of the orbit is 3, this vertex may be a local minimum [000] or a singular slope [100]. When the degree of the orbit is 4, the vertex may also be a slope [1001], and from an orbit degree of 5 upwards, this vertex may also be a saddle [01001]. Analogously, the vertex corresponding to the face maximum has to contain at least two consecutive 1-bits, [...11...], and based on the orbit degree it may be a local maximum, singular slope, slope, or saddle.

Proposition 2 *Let there be a vertex in the primal graph, which is a face minimum or a face maximum. By removing a redundant edge that is incident to this vertex, the LBP class of this vertex may only change, when its LBP class prior to the removal was a slope.*

Proof of Prop. 2 *Prior to the removal of a redundant edge, the orbit of the vertex corresponding to the face maximum contains at least 2 consecutive 1-bits, and the face minimum at least 2 consecutive 0-bits. By removing a redundant edge, this sequence of 0- or 1-bits gets shortened by 1 LBP value. This removal thus does not affect the number of* $0/1$ *and* $1/0$ *transitions, and thus does not change the LBP class if the vertex was a local minimum, local maximum, or a saddle. If the vertex was a singular slope, then only the multiple occurring 0- or 1-bits may be shortened, and thus not changing the LBP class. Finally, if the vertex was a slope, then it may only change its LBP class, when a shortening of a 0- or 1-bit sequence with length* 2 *occurs. Assume a slope of degree 4, i.e.* [0011]. *By removing a redundant edge, only the two LBP codes* [011] *or* [001] *may remain, and thus defining a singular slopes in both cases.* ■

In the dual graph, the removal of a redundant edge corresponds to a contraction of two vertices, where one of them is a singular slope. The only constraint here is, that the contraction is performed at the edge, that contains the single occurring 0- or 1-bit of the singular slope. This single occurring 0- or 1-bit of the singular slope is always paired with an opposite bit of the other vertex. By contracting the two vertices, the opposite bit gets replaced by n bits of the same "polarity" from the singular slope.

Proposition 3 *Let a redundant edge prior to its removal separate two faces, where one of them is a singular slope, and the other a saddle or a slope. By removing the redundant edge, the LBP class of the new face will be preserved, and thus will be a again a saddle or a slope.*

Proof of Prop. 3 *Consider a merging of a saddle* [0101] *and a singular slope* [0111] *in the dual graph. The single occurring 0-bit of the singular slope is paired with a 1-bit of the saddle. A contraction causes the paired edge to be removed and the remaining 0- and 1-bit sequences of both vertices to be combined at the point of merging. This way the previously existing 1-bit of the saddle is replaced by 3 1-bits of the singular slope, resulting in the LBP code* [011101] *or* [010111], *which is again a saddle. Analogously, the slope* [0011] *can only be extended by the 3 1-bits of the singular slope to the new LBP code* [001111], *which is again a slope.* ■

Proposition 4 *Let a redundant edge prior to its removal separate two faces, where both of them are singular slopes. By removing the redundant edge, the LBP class of the new face will be a singular slope or a slope.*

Proof of Prop. 4 *A constraint on the merging of two singular slopes is, that at least one of them has to be merged at the edge of the single occurrence of a 0- or 1-bit. Consider*

this first singular slope to have the LBP code [0111]*. Then the other singular slope has to be paired with the single 0-bit, that means it can have the LBP code* [0111] *or* [1000]*. In the first case, any 1-bit of the second singular slope is replaced by 3 1-bits of the first singular slope, and the resulting LBP code is* [011111]*, which is a singular slope. In the second case, both singular slopes are merged at their single 0- or 1-bits, and the resulting LBP code is* [000111]*, which is a slope.* ■

As can be seen, the only changes in the LBP classes in the primal and dual graph are between slopes and singular slopes. One important aspect of this observation is, that this removal of redundant edges can be chained together to completely remove singular slopes in the dual graph, such that the only remaining LBP classes in the dual graph will be slopes and saddles. As will be seen in the next section, it is also possible to completely remove saddles, with the exception of the outer face, such that only slopes remain in the dual graph. This way the whole dual graph will only consist of slopes, and thus describe all structural information within a face as two directed paths between a face minimum and face maximum.

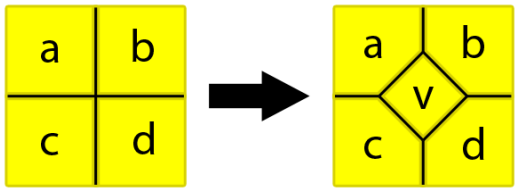
Another observation is, that the representative image $REP(I)$ is identical, when it is computed from the combinatorial map representing I before and after removal of redundant darts. This is due to the fact, that Algorithm 4.1 for computing the representative image computes the longest monotonically increasing paths between two reachable vertices. Since the removal of a redundant edge preserves the longer path inside a face that is a singular slope, this removal has no effect on the representative image.

4.3 Removal of Dual Saddles

In this section we will describe a method proposed in [GDKCL14] to remove dual saddles in an image. By applying this method, then merging plateaus, and then removing all redundant edges, it is possible to obtain an image representation, where except for the outer face, each face is a slope.

Prior to the merging of plateaus, with exception of the outer face, every vertex in the dual graph has the degree 4. In case of the outer face, the degree of the vertex is $2(M + N - 2)$, where M and N are the image dimensions. Saddles, that are inner faces, can be identified very easily by looking at the number of 0/1 and 1/0 transitions in the orbit of the dual vertex representing the face. Since the LBP code of this saddle can only be [0101] or [1010], the number of 0/1 and 1/0 transitions is 2 from each. This also means, that inside a face that is a saddle, there are 2 face minima and 2 face maxima. This is the only LBP class of faces, that has this characteristic. For this reason, to transform a dual saddle into a slope or a singular slope, it is necessary to split up this face, so that each new face has only one face minimum and one face maximum. Additionally it is necessary to perform this operation without any loss of structural information.

This transformation of a face that is a saddle can be performed by dividing it into 4 sub-faces. In the original image this operation translates to the insertion of a new sub-pixel in the middle of the face:



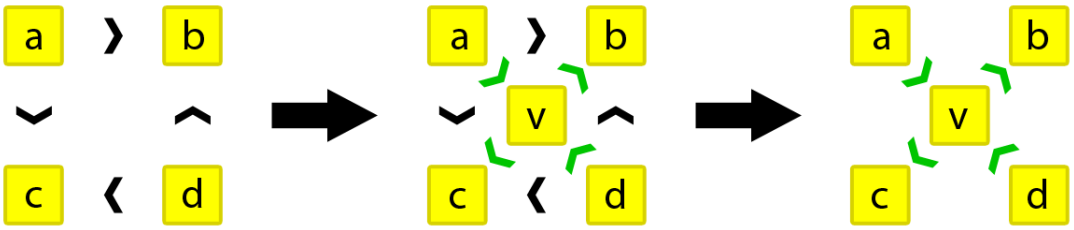
Let $g(x)$ be the grayscale value of a vertex x. Then the choice of the new grayscale value $g(v)$ of the vertex v inside the face depends on the relationships between the individual face minima and face maxima. Let a and d be the face minima, b and c the face maxima, and $g(a), g(b), g(c)$, and $g(d)$ the corresponding grayscale values. Then there are two possibilities to choose the value $g(v)$, namely:

1. **if** $g(a) = g(d) < g(b) = g(c)$ (**resp.** $g(a) < g(d) < g(b) < g(c)$)**, then** $g(v) = \frac{g(d)+g(b)}{2}$

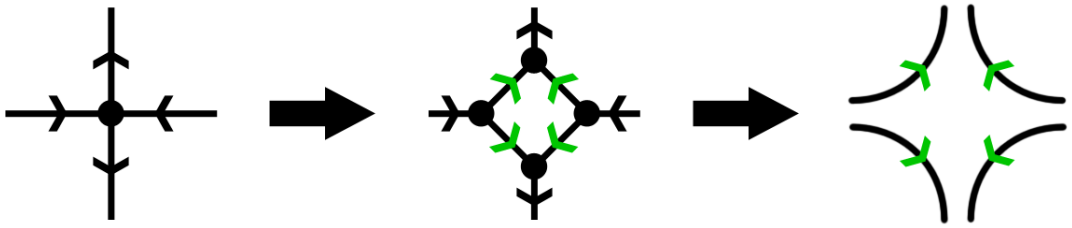
2. **if** $g(a) = g(d) < g(b) < g(c)$ (**resp.** $g(a) < g(d) < g(b) = g(c)$)**, then** $g(v) = g(a) = g(d)$ (**resp.** $g(v) = g(b) = g(c)$)

In the first case, either both diagonal pairs are equal, or none. Here it is important that the inserted vertex receives a grayscale value $g(v)$, which preserves the face minima and face maxima. Therefore the new grayscale value is in between the interval of the face minimum with the higher value, and the face maximum with the lower value. In the second case, only one of the diagonal pairs is equal. This situation suggests, that in between the diagonal pair is a connection, and thus the new vertex and the diagonal pair should form a plateau. Notice, that at this point no structural information is added or lost. The structural relationships between the face minima and face maxima remain the same, because it is still unknown which of the two face minima is structurally smaller, or which of the two face maxima is structurally greater.

This insertion of a new vertex causes the original face of degree 4, to be split into 4 new faces of degree 3. Since the face minima and face maxima remain preserved, the only two possible LBP codes of the new faces may be [001] and [011], which are singular slopes. The single occurring 0- or 1-bits correspond to the bounding edges of the original face. This way, after merging plateaus, it is possible to remove these edges, because they describe redundant binary relationships. An illustration of this insertion of a new vertex and subsequent removal of redundant edges, and the effects on the primal and dual graph can be seen in Figure 4.3.



(a)



(b)

Figure 4.3: Effects of inserting a new vertex into a face that is a saddle, and subsequently removing redundant edges. These effects are shown both on the (a) primal-, and the (b) dual graph. Notice in the dual graph on the right side, that no vertices are drawn. This is because the removal of an edge in the primal graph corresponds to a contraction in the dual graph, and all of the vertices in the dual graph were contracted towards outside.

4.4 Structural Correctness

All of the explained concepts in the previous sections have their existing equivalents in the order- and graph theory, and in this section we will make a connection between them.

Definition 11. Non-strictly Partially Ordered Set

A non-strictly partially ordered set is a set P with a binary relationship $R_P \subseteq P \times P$, such that for all $x, y, z \in P$:

- $(x, x) \in R_P$ *(reflexivity)*,
- $(x, y) \in R_P \wedge (y, x) \in R_P \Rightarrow x = y$ *(anti-symmetry)*, and
- $(x, y) \in R_P \wedge (y, z) \in R_P \Rightarrow (x, z) \in R_P$ *(transitivity)*.

Definition 12. Strictly Partially Ordered Set

A strictly partially ordered set is a set SP with a binary relationship $R_{SP} \subseteq SP \times SP$, such that for all $x, y, z \in SP$:

- $(x, x) \notin R_{SP}$ *(irreflexivity)*,
- $(x, y) \in R_{SP} \Rightarrow (y, x) \notin R_{SP}$ *(asymmetry)*, and
- $(x, y) \in R_{SP} \wedge (y, z) \in R_{SP} \Rightarrow (x, z) \in R_{SP}$ *(transitivity)*.

In analogy with a set of integers, the relationship $(x, y) \in R_P$ can be rewritten as $x \leq y$ (or equivalently, $x \geq y$), and the relationship $(x, y) \in R_{SP}$ can be rewritten as $x < y$ (or equivalently, $x > y$). Every non-strict partial order can induce a strict partial order through $(x, y) \in R_P \wedge x \neq y$. Similarly, every strict partial order can induce a non-strict partial order through $(x, y) \in R_{SP} \vee x = y$.

Definition 13. Non-strictly Totally Ordered Set

A non-strictly partially ordered set P is called a non-strict totally ordered set, if all of its elements are comparable, i.e. $(x, y) \in R_P \vee (y, x) \in R_P, \ \forall x, y \in P$.

Definition 14. Strictly Totally Ordered Set

A strictly partially ordered set SP is called a strict totally ordered set, if all of its elements are comparable, i.e. $(x, y) \in R_{SP} \vee (y, x) \in R_{SP}, \ \forall x, y \in SP$.

Definition 15. Cover

In a strictly partially ordered set SP, for $x, y, z \in SP$ the relation "x is covered by y" can be expressed as $(x, y) \in R_{SP}$ such there exists no element z with $(x, z) \in R_{SP} \wedge (z, y) \in R_{SP}$.

Having introduced the theory of partially and totally ordered sets, it is further possible to define an image and its structure.

Definition 16. Digital Grayscale Image

Let I be a non-strict total set of integers in the range $[0\ 255]$, and let there be some neighborhood definition between the elements of I, e.g. 4-, 6- or 8-neighborhood. Then by mapping this non-strict total set together with the neighborhood definition onto a grid, a grayscale image is formed.

Note however, that the individual pixels of the newly generated image do not form a total set anymore. The reason for this is, that not all of the pixels are connected through chains of "$\leq$" relationships. For example two local minima cannot be connected through such a chain, and thus they are not comparable. This way, the set of pixels of an image form merely a non-strictly partially ordered set.

Definition 17. Image Structure

Let P be the non-strictly partially ordered set of grid elements (pixels) of an image

induced by the order of the gray values. Then this set can also be interpreted as the structure of the image.

Notice, that with this definition we separate the image structure from the grayscale values of the pixels, and let it be defined purely by the "$\leq$" relationships mapped onto the grid.

When an image is represented as an attributed graph $G = (V, E)$, the vertices represent the pixels, the edges the neighborhood relationships, and the attributes stored at V the gray values. The image structure also defines a non-strict partial ordering of V. By merging pixels with equal values and removing self-loops in G a new reduced graph $G' = (V', E')$ is created, and a strict partial ordering of V' can be defined. This also allows to assign a direction $u \to v$ to each edge $(u, v) \in E$, turning G' into a directed acyclic graph (DAG).

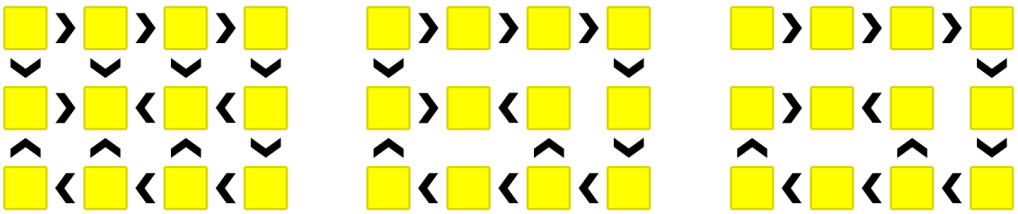
Definition 18. Minimum Equivalent Graph [MT69]
Let $G = (V, E)$ be a DAG. Then the minimum equivalent graph $G^* = (V, E^*)$ is the smallest sub-graph of G, such that there is a path from vertex u to vertex v in G^* whenever there is a path from u to v in G.

Definition 19. Transitive Reduction of a Graph [AGU72]
Let $G = (V, E)$ be a DAG. Then the transitive reduction $G^t = (V, E^t)$ is a graph with a minimum amount of edges, such that there is a path from vertex u to vertex v in G^t whenever there is a path from u to v in G.

Notice that the difference between the minimum equivalent graph G^* and the transitive reduction G^t is, that G^t does not necessarily need to be a sub-graph of the original graph G. One observation however is, that if the original graph is a planar DAG, then G^* and G^t are identical [AGU72]. Another observation by the authors is, that the computational complexity of the transitive reduction is equivalent to the computational complexity of performing a Boolean matrix multiplication. This computation can be done in polynomial time, and the currently fastest algorithm has the complexity $\mathcal{O}(n^{2.373})$ [Wil12].

A property of the minimum equivalent graph G^* is, that any edge in G^* may be contracted, and the strict partial ordering of the remaining vertices of the resulting graph remains preserved. This property is based on the fact, that the minimum equivalent graph is equal to the covering relation of the original graph [PH15]. What this means is, that if there is an edge between vertices u and v in G^*, then there does not exist a longer path between these two vertices in G^*. This way a contraction of an edge in the minimum equivalent graph does not produce any cycles.

As it is explained in [AGU72], the transitive reduction of a graph G can be obtained, by "*successively examining the arcs of G, in any order, and deleting those arcs which*



(a) G (b) G_{red} (c) $G^* = G^t$

Figure 4.4: An example where the reduced graph G_{red} is not equal to the minimum equivalent graph G^*. Vertices of the graphs are the yellow squares and directed edges are displayed as arrows. (a) is the original graph, (b) shows G_{red}, and (c) shows the minimum equivalent graph G^*, which is equal to the transitive reduction G^t.

are 'redundant', where an arc $\alpha = (u, v)$ is redundant if the graph contains a directed path from u to v which does not include α". From this algorithm description follows, that $G^* = G^t \subseteq G_{red} \subseteq G$, where G is the original DAG, G^* the minimum equivalent graph, G^t the transitive reduction, and G_{red} is the graph obtained by removing redundant edges from G, which correspond to single occurring 0- or 1-bits of dual singular slopes. The observation that G_{red} is not necessarily equal to G^t and G^* comes from the fact, that singular slopes stretching over multiple faces are not identified and this way some redundant edges may remain in G_{red}. An example of such a case, where $G^t = G^* \neq G_{red}$ is shown in Figure 4.4. The left image shows the original graph, the middle one shows G_{red} and the right one $G^t = G^*$. Notice, that in the middle image both faces are slopes, but when looking at the outer border, the LBP class is a singular slope.

An image segmentation is a subdivision of an image into a number of connected, non-overlapping, heterogeneous regions, where all pixels inside such a region share some homogeneous properties. These properties may be for example color, texture, some higher-level similarity measure, and so on. When an image segmentation is captured using a graph, then each vertex inside this graph corresponds to a region, and edges represent neighborhood relationships. In this work, we also define the direction of the edges, by mapping the image structure onto the graph capturing the image segmentation. This way smaller than ($<$), greater than ($>$), or greater or equal than ($\geq$) relationships are mapped onto the edges, and an ordering of the individual segmented regions is created. This allows us to define a *structurally correct image segmentation*:

Definition 20. Structurally correct image segmentation

An image segmentation is structurally correct, if the mapped structural relationships onto the segmentation regions satisfy all properties of a non-strict partial ordering. If additionally all regions with equal values have been merged, then a strict partial ordering has to apply.

CHAPTER 5

Structurally Correct Image Segmentation (SCIS) Algorithm

With the theory explained in the previous chapters we are able to propose a new structurally correct image segmentation algorithm. It is based on successively removing redundant edges and merging similar regions, while preserving all structural constraints. The main idea of this algorithm is that all the faces in the graph are slopes, and thus all of the structural information with respect to a face is captured as two directed paths between a face minimum and a face maximum. This way, if a merging operation causes a face to turn into a singular slope, redundant edges can be identified and removed, and new larger faces that are slopes are generated. In this work we represent an image as a graph with 4-connectivity, because a combinatorial pyramid is required to be planar, and it is not possible to construct planar graphs with 8-connectivity for images of size greater that 4×4 [KBL07]. This reduced neighborhood connectivity has however no effect on the LBP classes of individual regions if dual saddles are removed, as it is shown in [CGDK15].

Algorithm 5.1 describes the Structurally Correct Image Segmentation (SCIS) algorithm proposed in this work. The input of this algorithm is a 2-dimensional grayscale or color image I. From this image an attributed combinatorial pyramid in canonical representation is constructed in line 3. Each dart pair is assigned a weight, which represents the similarity between the associated vertices, $w(u_i, u_j) = |F(u_i) - F(u_j)|$, where u_i and u_j are vertices and F is some feature. For grayscale images F can be defined as the intensity of a region $F(u_i) = g(u_i)$, and for color images $F(u_i) = [r_i, g_i, b_i]$ for the RGB color distance, $F(u_i) = [v_i, v_i \cdot s_i \cdot sin(h_i), v_i \cdot s_i \cdot cos(h_i)]$ for the HSV color distance [SM00], or other features.

The Structurally Correct Image Segmentation (SCIS) algorithm consists of several methods:

Algorithm 5.1: Structurally Correct Image Segmentation (SCIS) Algorithm

Data: 2-dimensional digital image I
Result: A combinatorial pyramid in canonical representation capturing the segmentation process

```
1  k := 0;
2  Initialize vector J;
3  Initialize combinatorial pyramid C in canonical representation from image I;
4  C' := RemoveDualSaddles(C);
5  C_0 := MergePlateaus(C');
6  repeat
7  |  C'_k := RemoveRedundantEdges(C_k);
8  |  D := C'_k.getActiveDarts();
9  |  if D = ∅ then
10 |  |  return C'_k;
11 |  end
12 |  J := sortDartsByWeights(D);
13 |  while J.hasNext() do
14 |  |  d := J.getNext();
15 |  |  get vertices u and v associated to darts d and α(d);
16 |  |  val := computeNewValue(C'_k, u, v);
17 |  |  if fitsSurroundingLBP(C'_k, val) then
18 |  |  |  C_{k+1} := contractEdge(C'_k, d);
19 |  |  |  k := k + 1;
20 |  |  |  break while;
21 |  |  end
22 |  end
23 until C_k = C_{k-1};
```

- *RemoveDualSaddles*(): This method is based on Section 4.3. Here it is necessary to check the LBP class of each dual vertex, and process vertices that a saddles. In the combinatorial pyramid data structure this corresponds to traversing all orbits, thus the complexity is $\mathcal{O}(|\mathcal{D}|)$, which is equal to visiting each edge twice. The insertion of new vertices can be done by extending the α- and σ-permutation, and then relinking the σ-permutation. This operation needs to be done for each newly inserted vertex and is thus linear in the number of new vertices.

- *MergePlateaus*(): Plateaus can be identified by checking each edge, if the incident vertices have the same values. In the combinatorial pyramid data structure this relationship is encoded at the darts as binary attributes $l(d)$. Thus here it is sufficient to check for each pair of darts if $l(d) = l(\alpha(d)) = 1$ applies, thus the complexity is $\mathcal{O}(|E|)$.

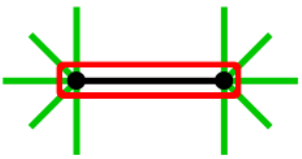
- *RemoveRedundantEdges*(): The removal of redundant edges can be performed

by computing the minimum equivalent graph of G_k. As was mentioned in the previous chapter, this operation has polynomial complexity, with the currently best performing algorithm having $\mathcal{O}(n^{2.373})$ [Wil12]. This part of the SCIS algorithm is the most expensive one, and in the next section we will show, that it is sufficient to remove redundant edges corresponding to single occurring 0- and 1-bits of dual saddles in combination with verifying that the new computed value fits all surrounding LBP values.

- $sortDartsByWeights()$: The sorting of the darts is necessary for the subsequent processing. The algorithm selects the dart with the lowest weight, and verifies if it can be merged. If it cannot be merged, then the dart with the second lowest weight is picked and verified. This is done until a suitable dart for merging is found, or else the algorithm stops. As both darts of an edge have equal weights, it is sufficient to select only the odd or even darts from the active darts. The sorting can be done in $\mathcal{O}(n \log n)$ in each iteration of the algorithm, where $n = |E| = |\mathcal{D}|/2$. Another approach is to store the edges in a sorted set, which allows multiple entries with the same values. After each merging, the edges incident to the new vertex are removed from the set, updated, and then reinserted. This way the complexity of filling the set in the beginning is $\mathcal{O}(n \log n)$, and then in each iteration the update of the set is $\mathcal{O}(2m \log n)$, where m is the number of updated elements, and in most cases $m \ll n$.

- $computeNewValue()$: The computation of the new value of the region depends on the application. This may be for example the mean or median of all the pixels in the region, or any other approach. In this work we utilize the mean operation. To speed up the computation, we additionally save the area of each region, and then when computing the new value, it is sufficient to compute the area-weighted mean of both regions.

- $fitsSurroundingLBP()$: This operation verifies, if the newly computed value satisfies all of the binary relationships stored at the incident edges. Since the two vertices that are candidates are not merged yet, this operation is equal to checking all the edges incident to the two vertices, minus the edges that connect them. In the following image, the edges that need to be checked are marked with green color:



In the combinatorial pyramid data structure this operation corresponds to checking two orbits minus the α-connected darts of the candidate edge, i.e. $\sigma^*(d) \setminus d$ and $\sigma^*(\alpha(d)) \setminus \alpha(d)$.

- *contractEdge*(): The contraction of the edge is implemented based on Section 2.2, and includes the modification of the σ-permutation and an update of the stored attributes.

The part of algorithm with the variable complexity is in the nested $while - loop$ in lines $13 - 22$. Here the algorithm selects edges with ascending weights and verifies if they satisfy the surrounding structural constraints, until a fitting one is found. One observation here is, that the complexity of this part depends on the amount of texture in an image. Images with little texture find fitting edges very early, and highly textured images have to verify a higher number of edges. This way images with a lot of texture require more time for processing. In the best case, where after merging plateaus only regions with non-equal values remain, the complexity of the outer and inner loop is combined $\mathcal{O}(|V| - 1)$, because in each iteration the first edge is selected, and the final equivalent contraction kernel is a tree. In the worst case, the last edge is selected, and therefore the complexity is $\mathcal{O}(\sum_{i=1}^{N} |E_i|)$, where N is again $|V| - 1$, and E_i is the set of edges at level i after redundant edges have been removed. In case all the pixels in the image have equal values, the algorithm terminates at line 10. The run-time of the algorithm applied to an artificial image with Gaussian noise and successively blurred copies of the image is shown in Section 5.2.

5.1 Redundant Edges and Surrounding LBP

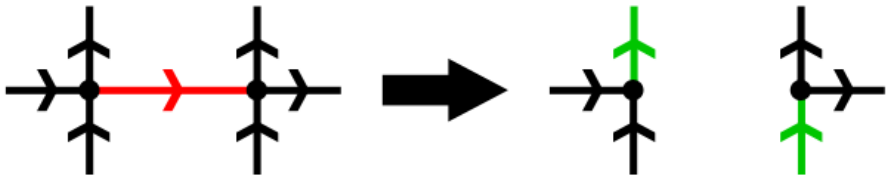
A property of a redundant edge is, that neither of the incident vertices is covered by the other. This means, that between two vertices u and v there is always at least one vertex w, such that $u < w < v$. Because of this property, there does not exist an interval of possible new values when u and v are merged, as the merged region would have to receive a new value that is at the same time greater and smaller than w. From this follows, that if not all redundant edges get removed in line 7 of the SCIS algorithm, they will be skipped in lines $16 - 17$. This way, instead of computing the minimum equivalent graph G^*, it is sufficient to just compute the graph G_{red}.

Proposition 5 *The SCIS Algorithm preserves the structural correctness of the image.*

Proof of Prop. 5 *The SCIS Algorithm computes the reduced graph G_{red}, and skips during contraction any edges included in $G^* \backslash G_{red}$, effectively allowing only contraction of edges of the minimum equivalent graph G^*. A contraction of any edge in G^* does not produce any cycles, see Section 4.4, and thus the strict partial ordering of the regions of the image is preserved.* ■

In the first iteration of SCIS, all of the redundant edges are present in C_0, and the *RemoveRedundantEdges()* function has to check each orbit in the dual graph if it is a singular slope. From this follows a complexity of $\mathcal{O}(|\mathcal{D}|)$ in the first iteration. Then in each subsequent iteration it is sufficient to check, if the merged edge did not create

any faces, that are singular slopes. Since a contraction operation in the primal graph is equivalent to a removal in the dual graph, only the incident faces need to be analyzed:



In this image the dual graph before and after merging two primal vertices is shown. The vertices in the image represent the incident faces of the edge that was contracted. A contraction in the primal is a removal in the dual, and thus the red edge is removed in the graph on the left. On the right, this dual removal caused both dual vertices to change LBP classes from slopes to singular slopes, and thus the edges marked with green color are redundant edges, and can be removed in the primal graph. As can be seen, the verification if new singular slopes were created can be reduced to $\mathcal{O}(|\varphi^*(d) \setminus d| + |\varphi^*(\alpha(d)) \setminus \alpha(d)|)$, where d and $\alpha(d)$ are the darts representing the contracted edge in the previous iteration.

5.2 Complexity with Amount of Texture

As was mentioned earlier, the part of the algorithm with the variable complexity is in the nested while-loop in lines $13 - 22$. Here the algorithm iterates over the edges in ascending order of their weights and picks the first edge, where the merging of the incident vertices satisfies the structural constraints of the local neighborhood. This approach suggests, that lowly textured regions get merged fast, and highly textured regions slowly. In this context, the "amount" of texture found locally around a region is defined by the variation of grayscale or color values of adjacent regions. This way, if the variation of grayscale or color values in the local neighborhood gets reduced, the "amount" of texture should decrease, and consequently the time to segment the image using the SCIS should decrease accordingly. To verify this, we let the algorithm run on a set of artificially generated images and measured the run-time. This set consists of an image with dimensions 300x300 and pixel values of 0.5 with additive Gaussian noise ($\mu = 0$, $\sigma = 0.01$), that is blurred with Gaussian filters of increasing variance. The noisy non-blurred image has a mean pixel value of 0.499, with the lowest value of 0.014 and the highest 0.999. During filtering, the border regions are handled by replicating the values of the other side. Figure 5.1 shows a plot of the results with the σ of the Gauss filters on the x-axis and the run-time in seconds on the y-axis. The non-filtered noisy image is excluded in the plot, and the run-time for this image was 2393 seconds. As expected, the run-time increases with the "amount" of texture in an image. In general the form of the run-time curve is exponential, with just some very slight deviations. These deviations can occur for example when the blurring caused some region configurations, so that less redundant edges could be removed, and thus more edges needed to be tested.

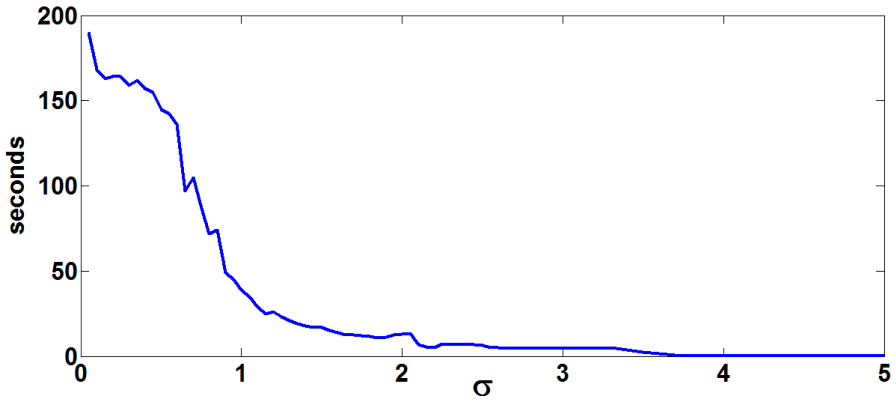


Figure 5.1: A plot showing the run-time of the SCIS algorithm based on the amount of texture in an image. A highly textured image with Gaussian noise was iteratively blurred with a Gauss filter of varying variance. The x-axis shows the variance of the filter, and the y-axis shows the run-time in seconds.

5.3 Region Merging History

An important property of the SCIS algorithm is, that highly textured regions remain preserved up to a high level in the pyramid. This is due to the fact, that these highly textured regions are composed of sub-regions which pose many structural constraints on each other. On one hand the edges between these sub-regions have rather high values and thus get selected as merging candidates by the algorithm in a late stage, and on the other hand the newly computed values by the algorithm very often do not satisfy the binary relationships stored at the incident edges. This way lowly textured regions are preferred by the algorithm for the merging in the lower levels of the pyramid. Figure 5.2 shows three natural images and their corresponding merging history. Dark regions in the "merging history image" were merged early, and bright regions were merged late. Notice in the top row, that the corn in the middle of the image, as well as the black beans on the right and the stripes of the bag get merged late, and that the regions that correspond to shadows or dirt get merged early on. Similarly, notice how in the middle row the sky and the dark areas get merged early, and how the highly textured crown of the tree and the bushes get preserved up to a high level of the pyramid. Also observe in the bottom row, that the latest merged regions are around the airplane, and then at the regions, where the sky blue and clouds get mixed.

Another observation here is, that these "merging history images" resemble images showing the magnitude of edges. In these images the magnitude is high at pixels, where the intensity in the local neighborhood changes abruptly, and low, where there are flat regions. Analogously, highly textured regions have a high variance in the intensity of the pixels in the local neighborhood and thus get preserved longer. This reflects in the "merging history image" as pixels with high intensity.

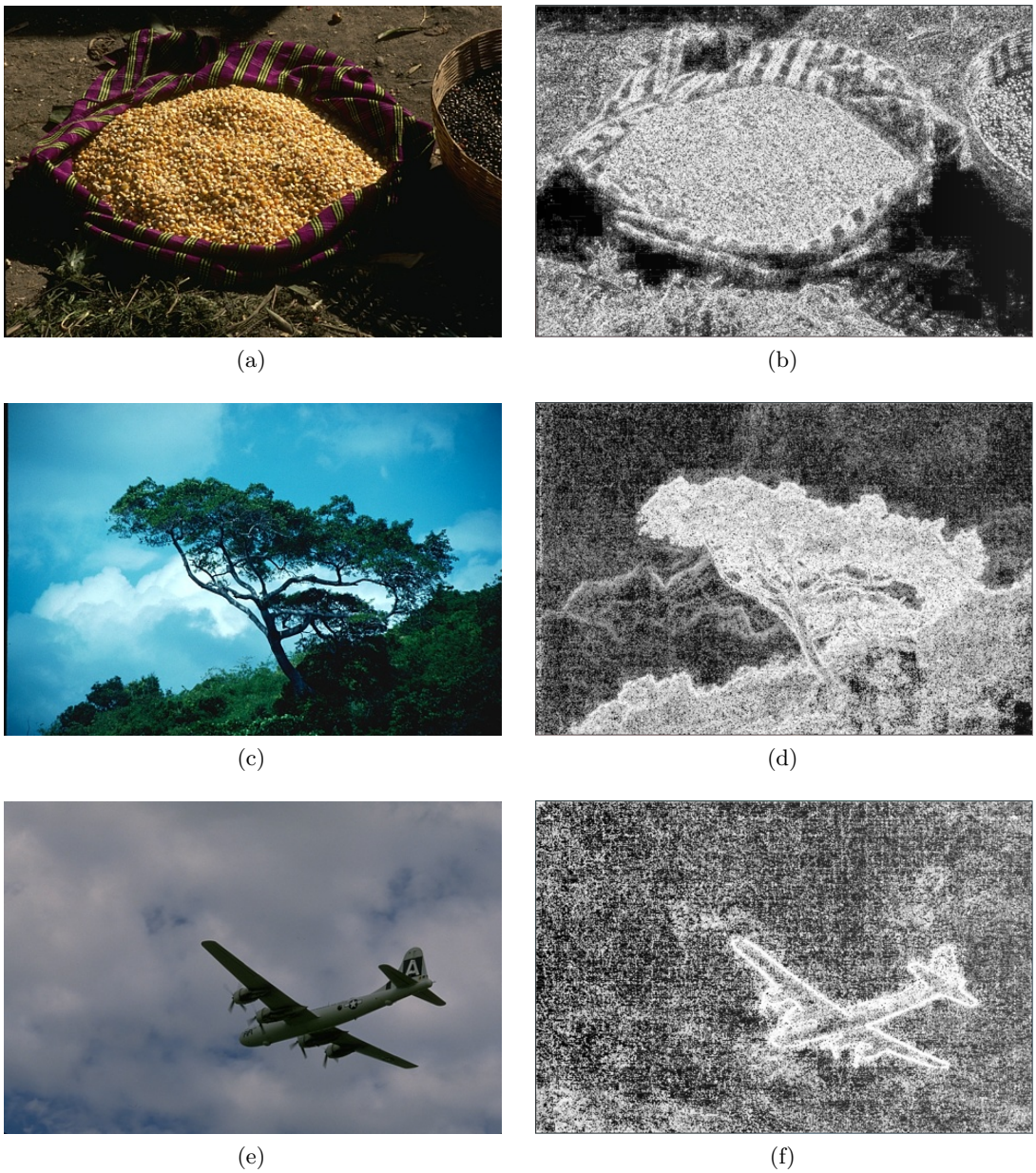


Figure 5.2: Merging history of the Structurally Correct Image Segmentation (SCIS) Algorithm shown on two example images. Images (a), (c), and (e) show the original images, and (b), (d), and (f) show the merging history. Darker regions were merged early by the algorithm, and bright regions were merged late.

Image Name	Min	Slope	Singular Slope	Saddle	Max	$\sum_{\text{pre}}$	$\sum_{\text{after}}$	$\frac{\sum_{\text{slopes}}}{\sum_{\text{after}}}$
220075.jpg	8299	60198	53201	15922	7997	165779	145617	0.778
210088.jpg	8432	65597	45708	16693	8710	167479	145140	0.766
106024.jpg	7864	46611	46147	14796	7764	164338	123182	0.753
12084.jpg	13779	51776	66488	26406	13045	174544	171494	0.689
108070.jpg	15808	44466	66788	30273	14880	177319	172215	0.646
119082.jpg	13112	33480	60637	25787	13364	167606	146380	0.643
14037.jpg	10525	28158	42172	19626	10243	167897	110724	0.635
145086.jpg	14849	40906	56727	28665	14446	175965	155593	0.627
102061.jpg	13315	27505	52649	25366	12948	170883	131783	0.608
101085.jpg	19065	33162	65904	37159	18638	182784	173928	0.569
156065.jpg	19033	34794	63424	37141	18724	183483	173116	0.567
196073.jpg	20239	26138	63783	40395	20761	185005	171316	0.524

Table 5.1: Topological class statistics for some randomly selected natural images from the Berkeley Image Segmenetation Dataset. The first column contains the names of the images, columns 2-6 show the individual topological classes, column 7 shows the number of regions after inserting new vertices into the image graph to remove dual saddles, column 8 shows the number of regions after merging plateaus, and the final column shows the percentage of slopes and singular slopes versus all regions.

5.4 Topological Classes in Images

In this section we show topological class statistics of some randomly selected natural images selected from the Berkeley Image Segmentation Dataset [MFTM01]. All of the shown images have dimensions 481×321 or 321×481, and thus the same number pixels, namely 154401. Each of these images has been stored in a combinatorial map, dual saddles have been removed by inserting new vertices, and plateaus have been merged, so that it is possible to classify each region into one of the topological classes. Table 5.1 shows the name of each of the selected images, the distribution of the topological classes, number of regions after inserting new vertices, number of regions after merging plateaus, and the percentage of combined slopes (slopes + singular slopes) from all topological classes. The statistics have been sorted descending by the percentage of combined slopes.

The main observation here is, that the number of singular slopes remains at a rather stable level. On the other hand, with increasing percentage of combined slopes, the number of local minima, local maxima, and saddles decreases, and the number of regular slopes increases. This behavior generally suggests, that images with a low percentage of combined slopes contain a high local variation in terms of grayscale or color values, and thus can be seen as rather highly textured. Such a high grayscale or color variation in the local neighborhood coupled with many local extrema cause pixel configurations, where it is often necessary to insert new vertices to remove dual saddles. This can be confirmed by looking at the overall number of vertices prior to and after plateau merging.

Analogously, images with a high percentage of combined slopes contain many local gradient-like features and less local variation, and thus can be seen as weakly textured. This observation can be verified by looking at the images in Figures 5.3 and 5.4.

The first figure shows the last three images from Table 5.1, which have a low combined slope percentage. As can be seen, these images contain large regions with high color variation. In the left image these regions are the three stone statues, the pebble on the ground below the statues, and the grass. In the top image these regions are the corals together with the overgrown wall on the right, and in the bottom image these regions are the sand and the snake. The second figure shows the first three images from Table 5.1. These images have a high combined slope percentage, which is caused by large blurred and gradient-like regions. The left image consists essentially only of gradient-like regions, namely the fish which is gradually shaded from orange to a dark yellow, and the corals which have a clean transition from light blue to dark blue color. The remaining two images on the right both are focused on the main objects, the cowboy and the penguin, which causes the objects in the background to be blurred, and thus have no fine texture.

(a) 101085.jpg

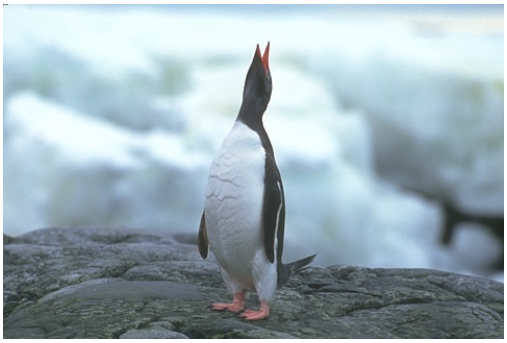
(b) 156065.jpg

(c) 196073.jpg

Figure 5.3: Highly textured images selected from the Berkeley Image Segmentation Dataset. Observe that these images contain rather high local variation in terms of color values, which causes it to contain many local extrema and saddles.

(a) 210088.jpg

(b) 220075.jpg



(c) 106024.jpg

Figure 5.4: Weakly textured images selected from the Berkeley Image Segmentation Dataset. Observe that these images contain little local variation in terms of color values, gradient-like regions, and some large blurred regions.

CHAPTER 6

Evaluation

In this chapter we will perform a quantitative evaluation of the Structurally Correct Image Segmentation (SCIS) algorithm and compare it to five other segmentation algorithms. The chosen dataset for the evaluation is the Berkeley Image Segmentation Dataset [MFTM01] which consists of 300 natural images of sizes 321×481 or 481×321 divided into a training set of 200 images and a test set of 100 images. Since none of the used algorithms for the evaluation require training, we have decided to use the 100 test images. The Berkeley Segmentation Dataset additionally provides ground-truth segmentation data produced by 30 human subjects, where each test image has at least 5 ground-truth segmentations. In these images, the main objects of interest are 22 human subjects, 43 animals, 10 man-made structures, 12 landscapes, and 13 other objects such as airplanes, cars, and so on. From a textural point of view, 22 images have little texture, 57 have medium amount of texture or an approximate ratio of lowly and highly textured regions of 1:1, and 21 images are highly textured. This way advantages and disadvantages of the texture preservation property of the SCIS algorithm can be analyzed and verified.

Image segmentation algorithms can be generally divided into two classes, namely boundary- and region-based methods [FMR+02]. The distinction between these two types is made based on the way pixels and their local neighborhood are handled. Boundary-based algorithms take the discontinuity between pixels and their neighbors into consideration, e.g. edge information, whereas region-based algorithms rely on the similarity between pixels. This similarity can be defined by some homogeneity property, such as similar colors, textures, etc. The SCIS algorithm falls into the region-based category, as the image is represented using a graph, and the merging is performed by finding the lowest weighted edge (highest similarity) which satisfies structural constraints. To compare the proposed SCIS algorithm, we selected four other region-based segmentation algorithms and one hybrid algorithm, which will be described in Section 6.2.

There are generally two types of image segmentation evaluation procedures:

- Qualitative, and
- Quantitative methods.

Qualitative methods rely on a subjective evaluation by human subjects. In this case the human subjects have to decide, whether the result of a segmentation algorithm is good or bad using some grading method. As this evaluation method is purely subjective and different observers provide different opinions and gradings, we focus on quantitative methods. Quantitative methods can be divided into analytical and empirical methods [Zha96]. Analytical methods evaluate the image segmentation algorithm in terms of its intrinsic properties, such as complexity, efficiency, etc. On the other hand, empirical methods evaluate the results of the algorithm, that means how close the computationally generated segmentation is to one or more reference segmentations. These reference segmentations are usually produced by humans in natural images, and a-priori defined in artificial images. The performance of the image segmentation algorithm using empirical methods is measured using evaluation functions. The analytical evaluation of the SCIS algorithm was performed in Sections 5.2 and 5.3. In this chapter, we will perform an empirical evaluation and comparison of the proposed SCIS algorithm and five other algorithms with respect to two well-known evaluation metrics, namely the *Global Consistency Error (GCE)* [MFTM01], and the *Probabilistic Rand Index (PRI)* [UPH07]. These metrics will be described in more detail in the next section. The characteristic of these two metrics is, that they are based on the overlap of the regions of two segmentations. We chose this type of evaluation metrics, because it coincides with the functionality of the region-based segmentation algorithms. In contrast, the boundary-based segmentation methods are usually evaluated using metrics, which are focused more on the dissimilarity properties of regions, such as *Boundary Precision-Recall Curves* [MFM04b] or the *F-Measure* [MFM04a].

The following chapter is divided as follows: Section 6.1 describes the Global Consistency Error and the Probabilistic Rand Index used for evaluating the computer generated segmentation to a set of ground-truth segmentations, and Section 6.2 provides an overview of the other five segmentation algorithms used in the evaluation. Section 6.3 shows the results of the evaluation and Section 6.4 discusses the advantages, disadvantages, and observations about the SCIS algorithm. Section 6.5 includes a gallery of some of the best and worst results, and finally Section 6.6 contains an overview of open problems and future work.

6.1 Metrics

In this section the Global Consistency Error and the Probabilistic Rand Index evaluation metrics will be described. Both of these metrics are based on the overlap of the regions of two segmentations, and the main advantage of these methods is that local refinement of segmentations is taken into account. In our evaluation we use ground-truth segmentations

on natural images produced by humans. As it is studied in [MFTM01], the segmentations produced by humans are generally consistent in segmenting individual objects, but the degree of granularity of the segmentations varies. For this reason local refinement is a factor which needs to be taken into consideration in segmentation evaluation. This granularity of segmentations can be also observed and thus studied in algorithms which provide a sequence of correlated segmentations, such as our proposed algorithm and several other we used for evaluation.

6.1.1 Global Consistency Error (GCE)

The Global Consistency Error (GCE) and the Local Consistency Error (LCE) were both introduced in [MFTM01]. Let $\setminus$ denote the set difference, and $|x|$ the cardinality of the set x. If $R(S, p_i)$ is the set of pixels corresponding to the region in segmentation S that contains pixel p_i, then the *local refinement error* is defined as

$$E(S_1, S_2, p_i) = \frac{|R(S_1, p_i) \setminus R(S_2, p_i)|}{|R(S_1, p_i)|}.$$

As can be seen, this local refinement error is not symmetric. This means that $E(S_1, S_2, p_i)$ is 0 only if S_1 is a refinement of S_2 at pixel p_i, but not vice versa. To provide a measure in both directions, the Global- and Local Consistency Errors are defined as

$$GCE(S_1, S_2) = \frac{1}{n} min \left\{ \sum_{p_i \in I} E(S_1, S_2, p_i), \sum_{p_i \in I} E(S_2, S_1, p_i) \right\}$$

and

$$LCE(S_1, S_2) = \frac{1}{n} \sum_{p_i \in I} min \left\{ E(S_1, S_2, p_i), E(S_2, S_1, p_i) \right\},$$

where n is the number of pixels in image I. As can be seen GCE is a "tougher" metric compared to LCE, because LCE allows refinement in both directions in a single region, whereas GCE allows only one direction per region. For this reason we chose to use only the GCE metric for our evaluations. These metrics have two extreme cases where the value is 0, namely when S_1 and S_2 are perfect overlaps, i.e. $S_1 = S_2$, or when S_1 is a perfect refinement of S_2, i.e. $S_1 \subsetneq S_2$, and vice versa. The second extreme case happens, if the number of regions in the segmentations produced by the segmentation algorithms is exactly 1.

6.1.2 Probabilistic Rand Index (PRI)

The original Rand Index [Ran71] was proposed by William Rand as a way to evaluate clustering methods. The main idea is to observe how pairs of points are placed into clusters in two different clusterings. Specifically, a similarity between two clusterings

exists, if the elements of an individual point-pair are placed together in a cluster in each of the two clusterings, or if they are assigned to different clusters in both clusterings. On the other hand, if the elements of a point-pair are placed in the same cluster in one clustering and in different clusters in the other clustering, then no similarity exists.

Let X be the set of N objects (or points) to be clustered, $X = \{x_1, x_2, ..., x_N\}$, and Y be a specific partitioning of these objects into K disjoint sets called clusters, $Y = \{Y_1, Y_2, ..., Y_K\}$. Each cluster is a set of the given points, $Y_k = \{x_{k_1}, x_{k_2}, ..., x_{k_{n_k}}\}$, with $\sum n_k = N$ and $n_k \geq 1$ for $k = 1, 2, ..., K$. Then the measure of similarity between two clusterings of the same data, Y and Y', can be defined as

$$c(Y, Y') = \frac{1}{\binom{N}{2}} \sum_{i<j}^{N} \gamma_{ij},$$

where

$$\gamma_{ij} = \begin{cases} \mathbf{1} & \text{if there exist } k \text{ and } k' \text{ such that both } x_i \text{ and } x_j \text{ are both in } Y_k \text{ and } Y'_{k'} \\ \mathbf{1} & \text{if there exist } k \text{ and } k' \text{ such that } x_i \text{ is both in } Y_k \text{ and } Y'_{k'} \text{ while } x_j \text{ is in} \\ & \text{neither } Y_k \text{ or } Y'_{k'} \\ \mathbf{0} & \text{otherwise} \end{cases}.$$

A property of the similarity measure $c(Y, Y')$ is, that $c = 0$ if the two clusterings Y and Y' have no similarities (i.e. one clustering consists of a single cluster and the other only of clusters containing single points), and $c = 1$ if the two clusterings are identical.

A segmentation of an image can also be interpreted as a clustering of the individual pixels. Here the set of pixels $P = \{p_1, p_2, ..., p_N\}$ corresponds to the set of points X, the segmentations S and S' to Y and Y', and the segmentation labels l_i and l'_i to Y_i and Y'_i. Then the previous equation can be rewritten in the context of measuring the similarity between two image segmentations as [UH05]:

$$RI(S, S') = \frac{1}{\binom{N}{2}} \sum_{i<j}^{N} [\, \mathbf{I}(l_i = l_j \ \wedge \ l'_i = l'_j) + \mathbf{I}(l_i \neq l_j \ \wedge \ l'_i \neq l'_j)],$$

where $\mathbf{I}$ is the identity function.

The Probabilistic Rand Index (PRI) [UPH07] is a generalization of the Rand Index and is able to compare one segmentation to multiple ground-truth segmentations. Let S be the computer generated segmentation that is compared with a manually labeled set of ground-truth segmentations $S_{\{1...K\}}$. The label of pixel p_i is denoted by l_i in

segmentation S and by l_i^k in the manually segmented image S_k. Then the Probabilistic Rand Index is defined as

$$PRI(S, S'_{\{1...K\}}) = \frac{1}{\binom{N}{2}} \sum_{i<j}^{N} [\ \mathbf{I}(l_i = l_j)P_{ij} + \mathbf{I}(l_i \neq l_j)(1 - P_{ij})],$$

where

$$P_{ij} = \frac{1}{K} \sum_{k=1}^{K} \mathbf{I}(l_i^k = l_j^k).$$

One characteristic of the PRI is that it does not penalize a refinement of a region, if there is support for this subdivision in the set of ground-truth segmentations. If however all segmentations in the ground-truth set agree on a region, then the refinement will be penalized. It is thus expected that this measure correctly evaluates a segmentation which is similar to different levels of granularity in the ground-truth segmentation set.

6.2 Algorithms

In this work we compared our proposed SCIS algorithm with five other algorithms. Four of these algorithms are, same as the SCIS algorithm, region-based, and one of them is a hybrid algorithm.

- *Greedy*: This algorithm is a simplification of the SCIS algorithm, where no redundant edges are removed and no structural constraints are respected. Same as the SCIS algorithm it models the image as a combinatorial pyramid, it performs contractions and simplifications, and preserves topology and the segmentation history of the image. The algorithm always selects non-incident edges with the lowest weights in the whole image and performs contractions, thus the *Greedy* denomination. This algorithm is good for a comparison of the isolated effect of the preservation of structural correctness in a segmentation algorithm. This algorithm has no parameters. As this algorithm is implemented using a combinatorial pyramid, a history of contractions is available. During an iteration multiple edges are contracted, and after each iteration the segmentation is evaluated. The results for each image are thus correlated.

- *IntExtMST*: This is an implementation of the algorithm proposed in [HK03] using combinatorial pyramids. This algorithm compares the internal and external contrast between regions to make decisions whether two regions are merged. The novelty of this algorithm is that edges are selected using Borůvka's algorithm [Bor26] and thus a Minimum Spanning Tree of the image is built. The only parameter in this algorithm is α, which sets preference on the size of the merged regions. In our

evaluation we use a fixed value of $\alpha = 1000$, as it is one of the two best performing α-values specified by the authors. Similarly to *Greedy*, this algorithm performs multiple contractions during one iteration. By using a combinatorial pyramid, the history of contractions is available and thus the segmentation results for each image are correlated.

- *Eff-Graph*: This is an algorithm proposed by Felzenszwalb and Huttenlocher [FH04], and an implementation of this algorithm from the authors is available on the internet. This is one of the first algorithms that use the principle of the internal and external contrast. The functionality is essentially the same as in *IntExtMST*, but without using Borůvka's algorithm. This algorithm has 3 parameters, namely σ which defines the size of a Gaussian filter, min which sets the minimal region size in a post-processing step, and α which sets the same preference on the size of the merged regions as in *IntExtMST*. Because we want to keep small details, we use default values of $\sigma = 0$ and $min = 10$. This algorithm provides no history of segmentations, just a final segmentation. To get segmentations with various numbers of regions we vary the last parameter α in a range $[0, 5000]$. The resulting segmentations are thus not correlated.

- *Mean-Shift*: This algorithm is based on a mean-shift segmentation approach presented in [CM02]. An implementation can be found on the internet in form of the application *EDISON*, which is an abbreviation for "*E*dge *D*etection and *I*mage *S*egmentati*ON* system". This algorithm performs a clustering in a 5-dimensional space, where 3 dimensions are the $L^*u^*v^*$ color components and the remaining 2 dimensions are the spatial coordinates. This algorithm has 3 parameters, namely h_s and h_r which define the kernel bandwidths in the spatial and color dimensions, and m which defines the minimal region size in a post-processing step. The result of this algorithm is a single segmentation corresponding to the parameters, thus the results with different region sizes are not correlated. We use the default value $m = 10$, and use equal for the kernel bandwidths. These kernel sizes are iteratively increased in an interval $[1, 15]$.

- *Turbopixels*: The last used algorithm for the evaluation is an algorithm based on the superpixel approach. The idea is to divide the image into a lattice-like structure of compact regions (superpixels) by iteratively dilating seeds which adapt to the local structure of the image. We use an implementation of the paper [LSK+09] available on the internet. This approach grows the individual regions in different directions at different speeds. The growing speed is defined by the proximity to other regions and the local image structure. This algorithm has only one parameter, namely n which defines the number of initial seed points and thus also the final number of regions in the segmentation. The results of this algorithm are not correlated. Even though this algorithm works on a completely different principle than all of the other algorithms used in the evaluation, we decided to include it because the effect of a completely different approach on the GCE and PRI metrics can be studied.

We evaluate and compare these algorithms both on color and grayscale versions of the test images. All three algorithms that we implemented (*SCIS, Greedy, IntExtMST*) utilize the same similarity measure to weight the edges, and the mean operator to generate new region values. In the case of grayscale images we use the absolute difference between region intensities, i.e. $w(u_i, u_j) = |g(u) - g(u_j)|$, where $g(u_i)$ and $g(u_j)$ are the intensities of regions u_i and u_j. Color images are transformed into the CIE $L^*a^*b^*$ color space and the edges are weighted using the perceptually uniform CIEDE2000 color difference [SWD05]. Using this approach we hope to get segmentation results closer to the human perception, than by simply using the RGB color space. The *Eff-Graph* algorithm weights edges in grayscale images using the absolute intensity difference. For color images the algorithm is run three times for each color channel in RGB color space. If two neighboring pixels appear in the same component in all three of the color channel segmentations, then they are put into the same component in the final segmentation, otherwise the pixels will be in two separate components. The *Mean-Shift* algorithm transforms all images into the $L^*u^*v^*$ color space corresponding to the first 3 dimensions. Grayscale images are handled as RGB images with all channels being equal to the grayscale image. Finally, the *Turbopixels* algorithm utilizes the grayscale image gradient to make decision how fast which region should grow in what direction. RGB images are being linearly transformed to grayscale images.

6.3 Results

For the visualization of the results we utilize the method proposed by Hanbury and Stöttinger in [HS08]. We run all six segmentation algorithms with the discussed parameters on the 100 test images of the Berkeley Image Segmentation Dataset [MFTM01], and plot the results on a graph with the number of regions on the x-axis and the GCE- resp. PRI error on the y-axis. An example of such a graph can be seen in Figure 6.1, where the results of the *Greedy* algorithm for color images are shown. Here each blue dot represents a segmentation result. Since only two of the total six segmentation algorithms are able to produce a segmentation with a specific amount of regions, namely the *SCIS* and the *Turbopixels* algorithms, the results have to be clustered. This is done by placing the results into a number of non-uniform sized bins. Regions where there is a high concentration of segmentation results and the variation along the y-axis is high get small bin sizes, and regions with a low concentration of segmentation results and low variation along the y-axis get larger bin sizes. The borders of these bins are illustrated in Figure 6.1 as red vertical lines. Then for each bin the mean number of regions and mean GCE- and PRI error is computed and plotted as a red colored star. Finally, to obtain the GCE- and PRI error curves, the mean GCE- and PRI errors for each bin are connected, marked as thick red lines in Figure 6.1.

Notice in Figure 6.1, the overall trend of the data is that at a high number of regions the variation along the y-axis is rather low, and at a low number of regions the GCE rises quadratically. Also notice that at the very left end the GCE falls back down to 0. This

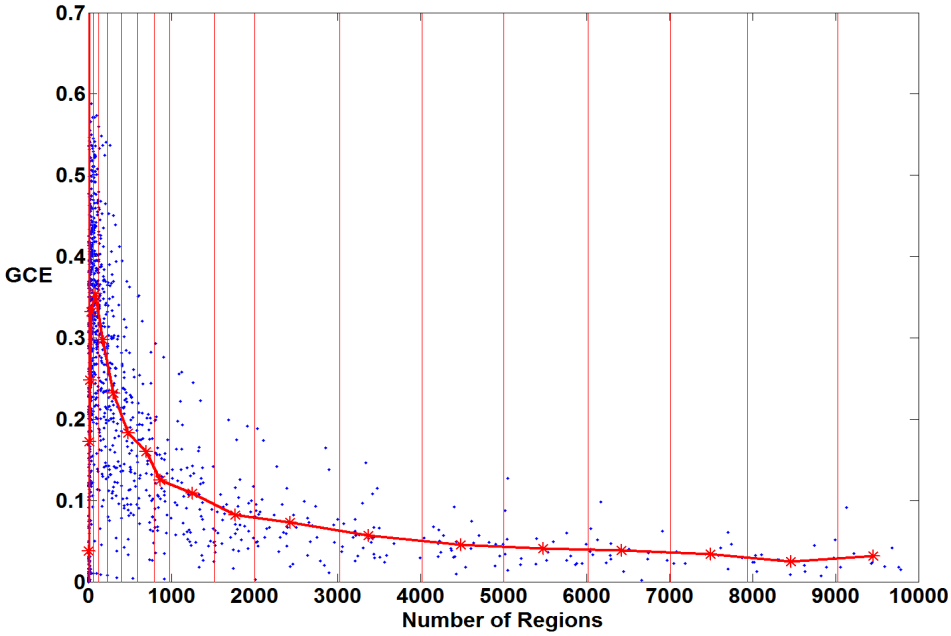


Figure 6.1: An example of generating the GCE error curve for color images of the *Greedy* algorithm. The segmentation results are marked as blue dots, the bin boundaries are marked as red vertical lines, the mean GCE error of each bin is marked as a red star, and the thick red line shows the GCE error curve.

is the second extreme case of the GCE we discussed in Section 6.1, where the number of regions in the computer generated segmentation is lower than in the ground-truth data, and the ground-truth data becomes a refinement of the other segmentation. All of the GCE error curves show the same form, even for the algorithms with uncorrelated data. For this reason, all GCE error curves can be evaluated by their maximum GCE, as well as by the form of the curve. *A good segmentation should have a low maximum GCE and the curve should be increasing in a slow fashion, the later (lower number of regions) the better.*

The form of the PRI error curves for all algorithms is also the same, namely a rather steady profile at above 500 regions, and then a fast decline towards 0 as the number of regions approaches 1. This decline is an expected result, as the PRI metric penalizes cases where two points in the same region in the computer generated segmentations should be in two different regions in the ground-truth data. *A good segmentation thus should have a high as possible PRI for as few regions as possible.*

To maintain a good readability and to be able to better focus on the comparative results, we include all plots with the exact segmentation results and the error curves in Appendix A. In this section we plot the GCE- and PRI error curves of all algorithms for grayscale and color images next to each other in corresponding plots. We divide each comparative plot into two plots, namely an overview of all the error curves in a region

interval of $[0, 10000]$, and a zoomed in plot at the region interval $[0, 700]$. The GCE plots for grayscale and color images can be seen in Figures 6.2 and 6.3 resp. Figures 6.6 and 6.7. The PRI plots for grayscale and color images can be seen in Figures 6.4 and 6.5 resp. Figures 6.8 and 6.9.

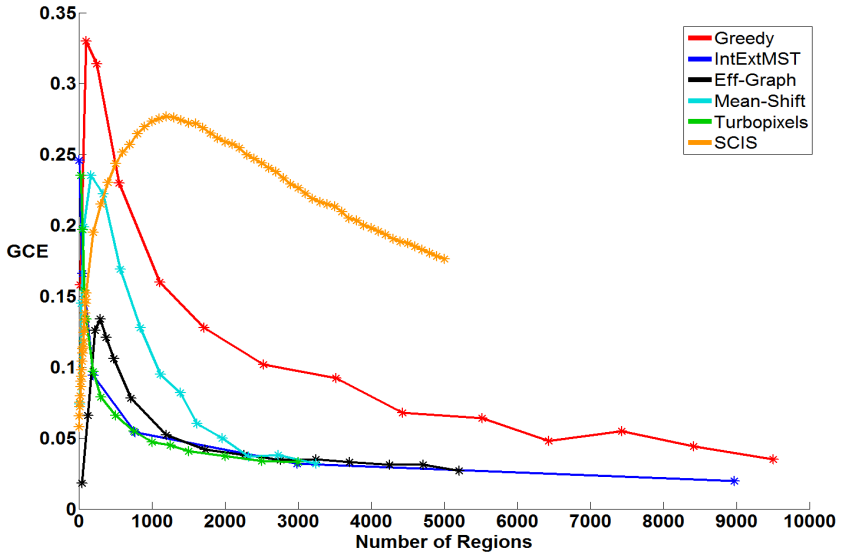


Figure 6.2: A comparative plot of all algorithms for the GCE metric for grayscale images for all regions.

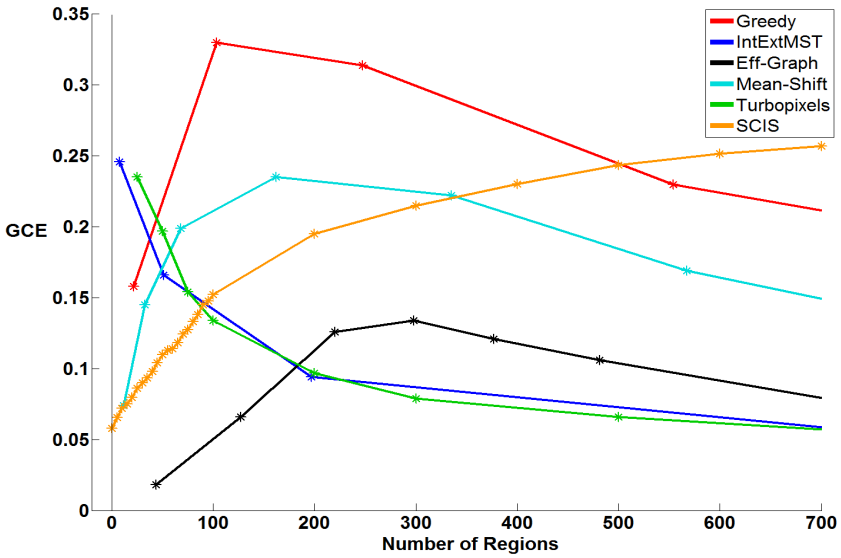


Figure 6.3: A zoomed in comparative plot of all algorithms for the GCE metric for grayscale images for regions in the interval $[0, 700]$.

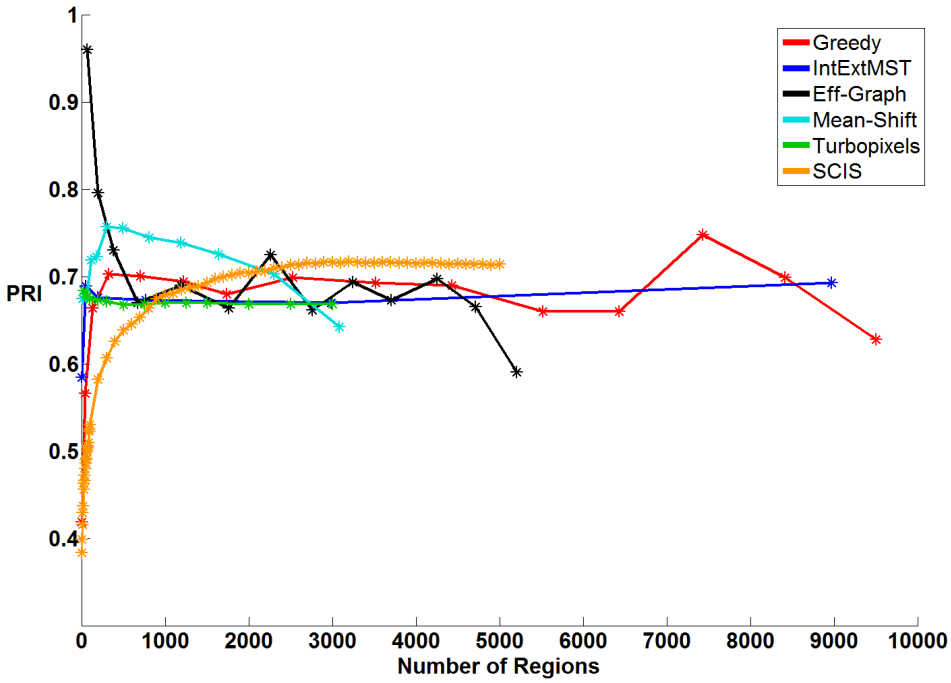


Figure 6.4: A comparative plot of all algorithms for the PRI metric for grayscale images for all regions.

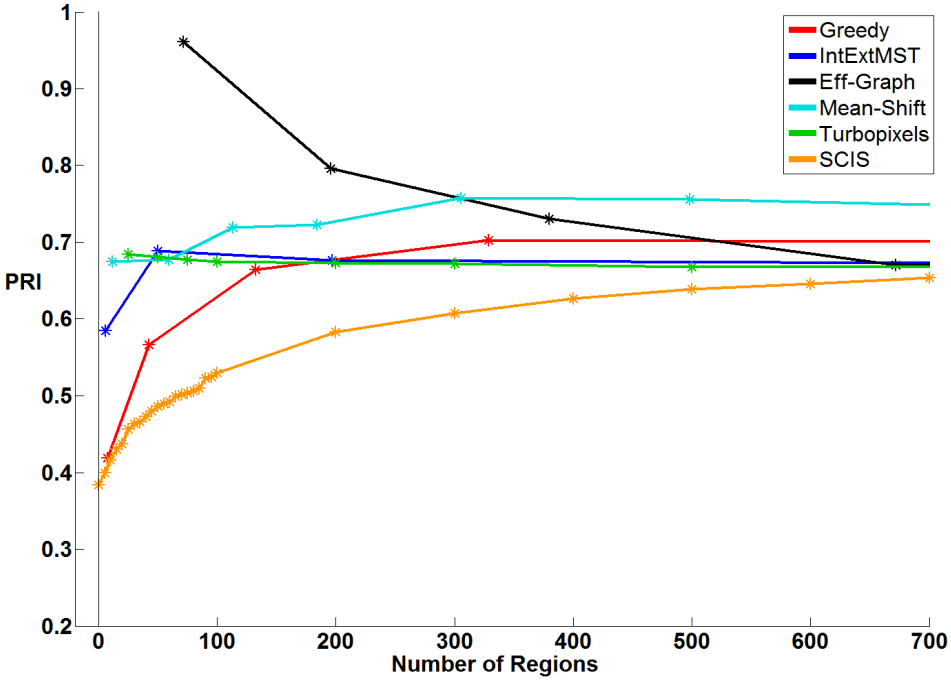


Figure 6.5: A zoomed in comparative plot of all algorithms for the PRI metric for grayscale images for regions in the interval $[0, 700]$.

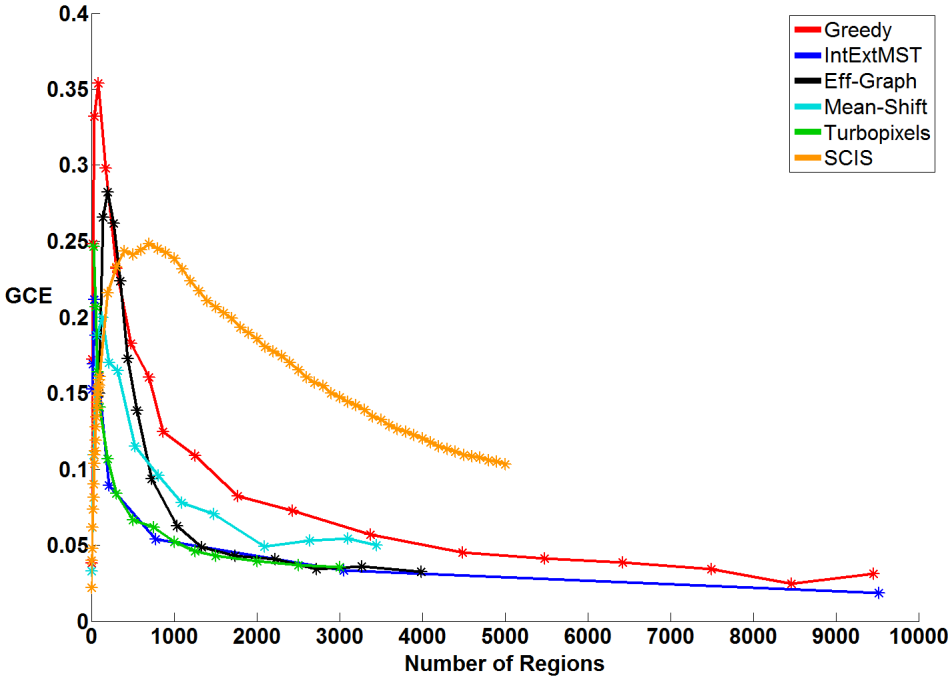


Figure 6.6: A comparative plot of all algorithms for the GCE metric for color images for all regions.

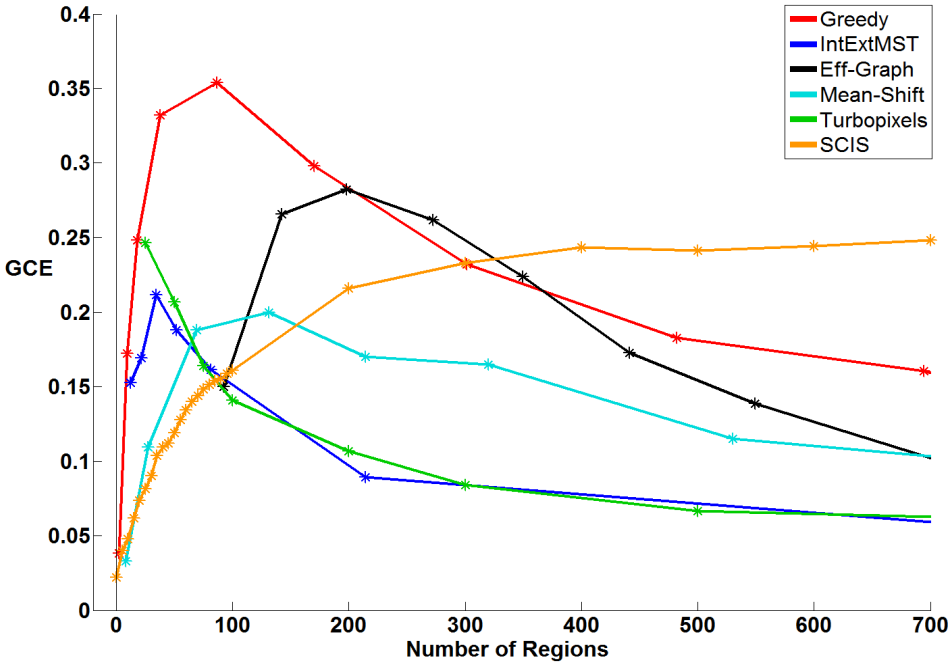


Figure 6.7: A zoomed in comparative plot of all algorithms for the GCE metric for color images for regions in the interval $[0, 700]$.

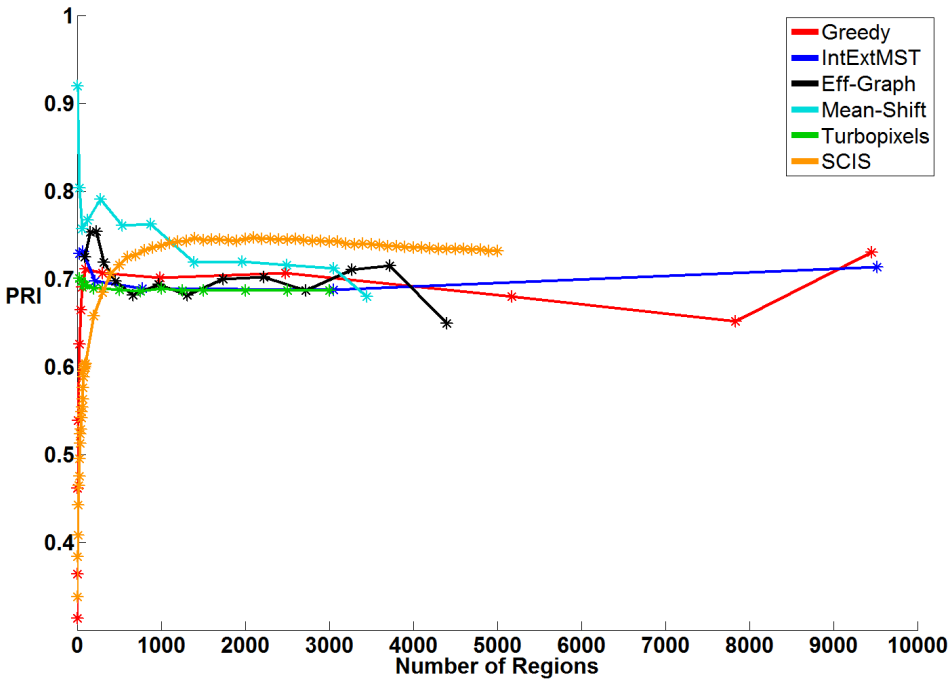


Figure 6.8: A comparative plot of all algorithms for the PRI metric for color images for all regions.

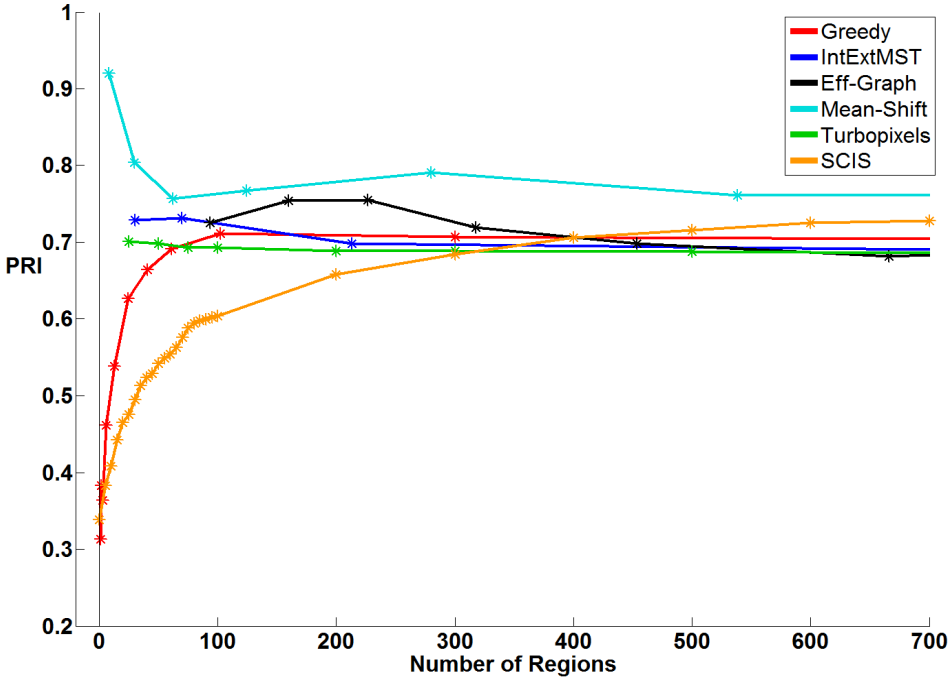


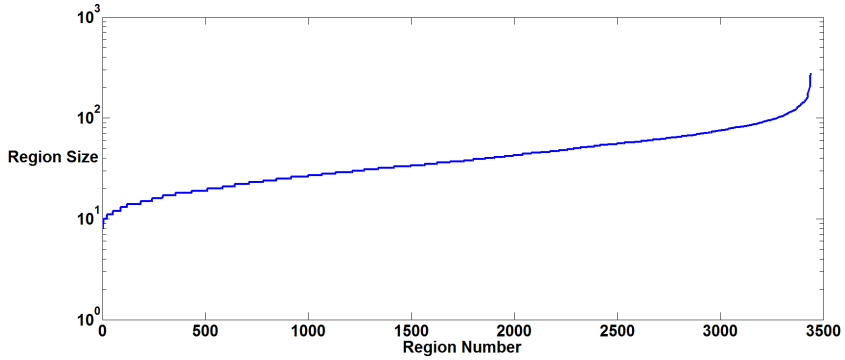
Figure 6.9: A zoomed in comparative plot of all algorithms for the PRI metric for color images for regions in the interval $[0, 700]$.

6.4 Discussion

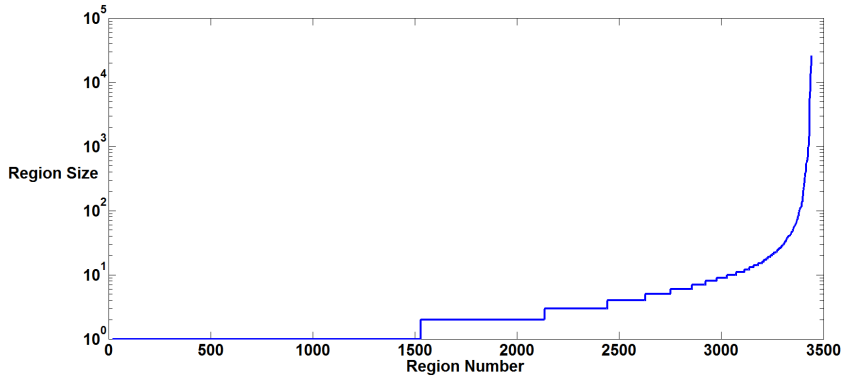
One of the main observations about the *SCIS* algorithm is, that the whole GCE error curve is shifted more to the right and the incline is much slower, when compared to all other segmentation algorithms. This bad performance comes from the texture preserving property of the *SCIS* algorithm, because weakly textured regions are undersegmented in favor of preserving highly textured regions. Here the highly textured regions consist of a high number of very small connected components, whereas the weakly textured regions are merged very early and generally consist of a few large connected components. These large components then generate a high GCE error, because they stretch over multiple smaller components in the ground-truth segmentations. To verify this observation, we compared the sizes of the regions of the *IntExtMST* and the *SCIS* algorithms at the same level of the pyramid. To give an example, Figure 6.10 shows the sorted region sizes of the *IntExtMST* and the *SCIS* algorithms for the image "43074.jpg" (shown in Figure 6.11a) at 3440 regions. Notice, that more than 90 percent of the regions in the *IntExtMST* algorithm have a size in the interval $[10, 100]$, whereas around 90 percent of the regions in the *SCIS* algorithm have a size in the interval $[1, 10]$. Then, the sizes of the regions in the remaining 10 percent of the *SCIS* sharply increase up to a maximum of more than 10000. These large regions are exactly the weakly textured regions, which are merged early in the segmentation process and also cause a high GCE error early on.

The maximum GCE error of the *SCIS* algorithm for grayscale images is the second worst. This bad performance can be explained by the lack of information, as the number of values in an 8-bit grayscale image is 256, and in a color image 256^3. The problem is, that a texture may consist of different colors that have similar brightness, and thus when the image is converted to grayscale, all of these colors receive nearly the same grayscale value. This way in a color image there is clear evidence for a texture, but in the grayscale image all of this structural information is much less accentuated. When comparing the maximum GCE error for color images, the *SCIS* algorithm provides better results than the *Greedy* and the *Eff-Graph* algorithm, and is on the same level as the *Turbopixels* algorithm. In both the grayscale- and color images, the *Greedy* algorithm provides the worst performance with respect to the maximum GCE error. This result is expected, as this algorithm chooses simply the edge with the lowest weight and performs a contraction on it. No assumptions regarding redundant edges, internal- or external contrast, or structural correctness are being made. Adding structural constraints to the segmentation process provides clearly an improvement, as can be seen when compared to the *SCIS* algorithm.

The same observation about the right shift of the GCE error curve also applies to the PRI error curve. However, in the case of the PRI error metric, the *SCIS* excels above all algorithm in terms of a steady high PRI score. For grayscale images, the PRI error curve starts falling off at around 2500 regions, and the algorithm starts performing worse than all other algorithms at around 1500 regions. In case of color images, the PRI error curve starts falling off at around 1500 regions, and the algorithms performs worst at



(a)



(b)

Figure 6.10: Sorted region sizes for the image "43074.jpg" of the (a) *IntExtMST* and the (b) *SCIS* algorithm at 3440 regions. Notice, the graphs have a logarithmic y-axis for a better overview.

around 350 regions. When considering the fact that most of the connected components at each level of the pyramid are in the interval $[1, 10]$, this means, that the segmentation happens at regions, where there is evidence for segmentations at different granularity levels produced by human subjects. This way the segmentations do not get penalized by the PRI error metric. This result suggests, that highly textured regions carry more visual information that is more important to us humans, than lowly textured regions. As a consequence, the SCIS segmentation algorithm may find its application in the field of image compression. On one hand side, the use of the canonical representation of the combinatorial pyramid allows a full reconstruction of the image from the apex at the storage costs of the original image. If the full merging history is not required, it is sufficient to only store a far more reduced attributed graph at the current level of the pyramid. On the other side, the algorithm performs a reduction in region size, while at the same time maintaining the visually important features of the image. During evaluation we observed, that around 70 percent of the regions in a medium textured image may be contracted, while perceptually observing only a minimal change in the image. For a test image from

the Berkeley Image Segmentation Database of size 481×321 = 154401 pixels this means, that it may be reduced to around 45000 regions, while keeping the most important information. To give an example, observe the two images shown in Figure 6.11. The top image shows the original image of size 481×321 and a total number of 154401 pixels, and the bottom image shows the segmentation result of the *SCIS* algorithm at 46000 regions. Notice, that the bird is perfectly preserved, as well as all of the dry grass. Only a change can be observed at the blurry green background, but perceptually this change plays only a minimal role. To show even a higher compression, observe Figures 6.12 and 6.13. The first figure shows a reduction using the *SCIS* algorithm to 15000 regions, which is a reduction of around 90 percent, and the second figure shows a reduction to 5000 regions, which is a reduction of around 97 percent. Observe, that the highly textured regions are still very well preserved. A comparison of the results of the *SCIS*- and the *IntExtMST* algorithms at 12904 regions (region reduction of 92 percent) can be seen in Figure 6.14. Notice how the highly textured wing and the dry grass in the foreground are preserved almost without a perceptual change by the *SCIS* algorithm in the top image, while the *IntExtMST* better preserves the rather "uninteresting" green background at the cost of destroying the fine texture of the bird and the grass. An obvious disadvantage of the *SCIS* is, that if the main object of interest in an image is less textured than the background, then it will be reduced early in the process, which may in some cases cause bad perceptive results.

The *Turbopixels* algorithm provides as expected good results in both metrics. This is given by the fact that this approach ignores any texture and purely looks for well defined edges, which in the Berkeley dataset clearly separate the main object from the "uninteresting" background. All of the regions have almost the same size and are evenly distributed over the whole image. This algorithm provides an even refinement of the ground-truth segmentations, which explains the good scores in the GCE metric. The PRI metric however penalizes a refinement of large regions and thus the lower PRI scores over the course of the whole segmentation. The *IntExtMST* and the *Eff-Graph* algorithms have a tendency to produce regions of almost the same size, but they are more centered around a texture of a region. In general the segmentations produced by both algorithms produce similar results like the *Turbopixels* algorithm and thus also provide similar results in both metrics. The *Mean-Shift* algorithm is based on a clustering approach in 5-dimensional space. A textured region can be seen in 3-dimensional color space as a cluster, where the variation along the individual axes depends on the degree of "texturedness". Blurred and low textured regions get merged, and highly textured regions get preserved to a certain degree. This would also explain the good scores in both metrics.

We implemented the *SCIS* algorithm using Visual C++ on a 32-bit Microsoft Windows platform. The average computation time for the test images of the Berkeley Image Segmentation Database of sizes 481×321 and 321×481 were on a 2.66 GHz Intel Core 2 computer 70 seconds for color images, and around 40 seconds for grayscale images. As was mentioned in Section 6.2, we used the CIEDE2000 distance for color images, and the absolute intensity difference for grayscale images.

(a)

(b)

Figure 6.11: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 46000 regions by the *SCIS* algorithm.

Figure 6.12: A reduced image to 15000 regions by the *SCIS* algorithm from Figure 6.11a.

Figure 6.13: A reduced image to 5000 regions by the *SCIS* algorithm from Figure 6.11a.

(a)

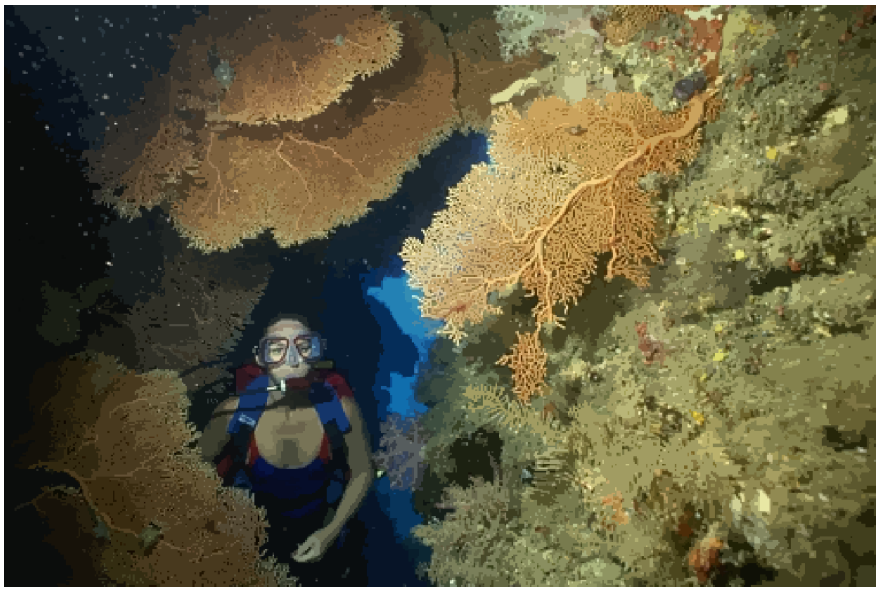
(b)

Figure 6.14: A comparison of the reduced images to 12904 regions by the (a) *SCIS*- and the (b) *IntExtMST* algorithm. The original image is shown in Figure 6.11a and has the size 481×321 and a total of 154401 pixels.

6.5 Gallery

6.5.1 Best Results

(a)



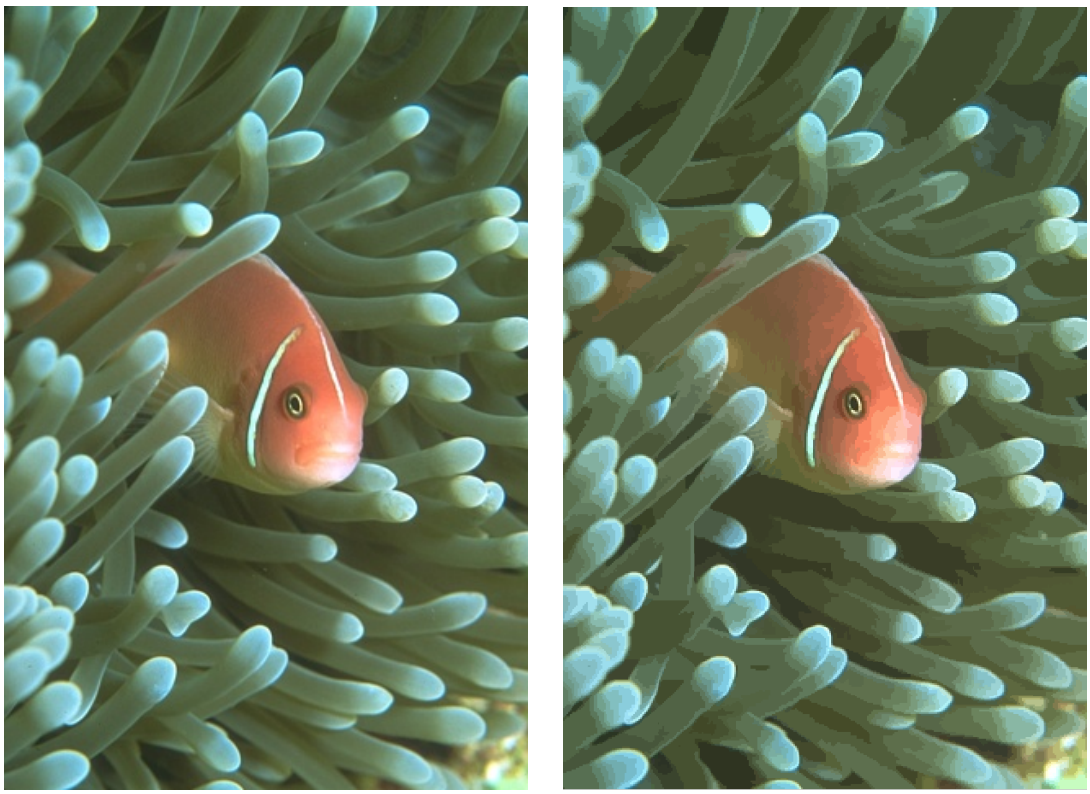
(b)

Figure 6.15: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 12352 regions by the *SCIS* algorithm, which is a reduction by 92 percent. Notice the well preserved fine texture of the coral in the middle.

(a)

(b)

Figure 6.16: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 9264 regions by the *SCIS* algorithm, which is a reduction by 94 percent. Even though the background is destroyed, notice the extremely well preserved fur of the animal and the bark.



(a) (b)

Figure 6.17: A comparison of an (a) original image of size 312×481 with 154401 regions and its (b) reduced version with 9264 regions by the *SCIS* algorithm, which is a reduction by 94 percent. This example shows how well generally low textured images are preserved.

(a)

(b)

Figure 6.18: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 9264 regions by the *SCIS* algorithm, which is a reduction by 94 percent. Notice how well the fur of the squirrel is preserved. Once the background is just a little bit blurry, the clear regions are preferred.

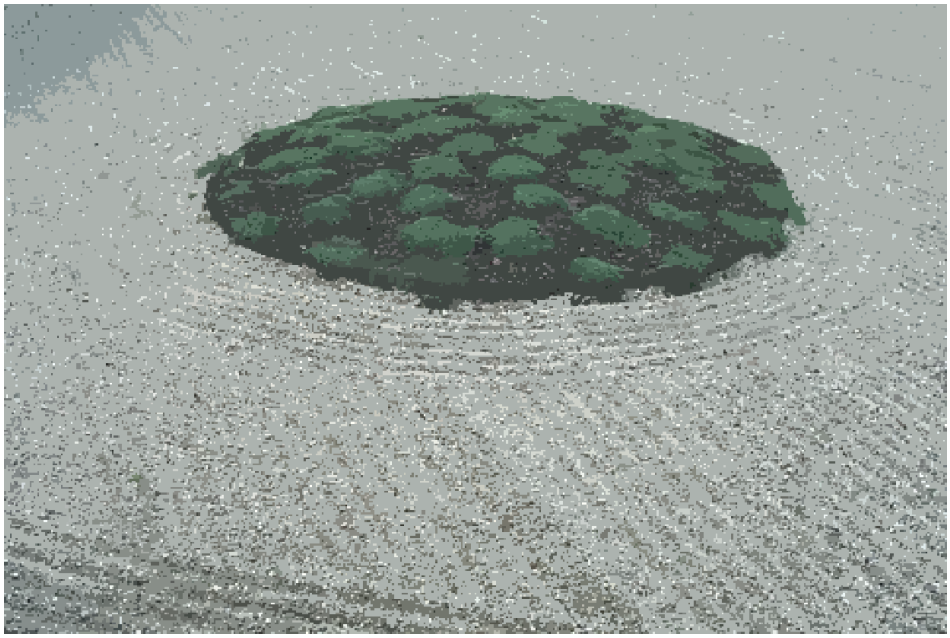
6.5.2 Worst Results

(a)

(b)

Figure 6.19: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 15440 regions by the *SCIS* algorithm, which is a reduction by 90 percent. Notice that the main object of interest, which is the animal, is less textured than the rocks. At this point regions of the animal are getting merged earlier than the background.

(a)



(b)

Figure 6.20: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 12352 regions by the *SCIS* algorithm, which is a reduction by 92 percent. Notice how the individual highlights of the rocks (local minima and maxima) are well preserved, even though they are rather uninteresting for the content of the image.

(a) (b)

Figure 6.21: A comparison of an (a) original image of size 312×481 with 154401 regions and its (b) reduced version with 15440 regions by the *SCIS* algorithm, which is a reduction by 90 percent. Notice how the stone statue is merged together with the ground, even though they have rather different colors.

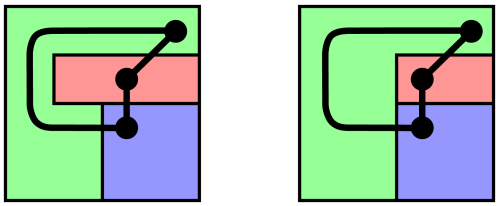
(a)

(b)

Figure 6.22: A comparison of an (a) original image of size 481×312 with 154401 regions and its (b) reduced version with 15440 regions by the *SCIS* algorithm, which is a reduction by 90 percent. Notice the step-effect of the *SCIS* algorithm on smooth gradients like the sky in the image.

6.6 Future Work

In Section 6.4 we proposed to use the SCIS algorithm in the field of image compression due to its texture preserving properties. These properties allow a high reduction in region number, while keeping the perceptively important information of an image, encoded as textured regions, intact. There is however a distinction to be made between reduction in region number, and an overall reduction in terms of storage costs. Currently, the whole merging history of the combinatorial pyramid can be stored at the storage costs of the base combinatorial map with the canonical encoding scheme. Dependent on the size of the currently active set of darts, the combinatorial map representing the current partition of the image changes in size. As it was explained in Section 6.4, it is possible to merge around 70 percent of all regions in a medium textured image, while perceptually observing only a minimal change in the image. At this point it would be possible to "cut away" the passive part of the canonical combinatorial pyramid, and store the "reduced" combinatorial pyramid at lower storage costs. If some compression artifacts are allowed, then it would be possible to "cut away" the pyramid at an even higher level. The problem however is, that even though the "reduced" combinatorial pyramid uniquely stores the structure of the image in terms of adjacency and inclusion relationships of regions, it is still necessary to store the mapping of regions onto the image grid. Observe the following two partitions of an image:



As can be seen, both images have the same dimensions and are divided into three non-overlapping regions. The combinatorial maps encoding the neighborhood relationships of the regions are also identical. Without a mapping of the regions onto the image grid, it would not be possible to deduce which of these two image the combinatorial map is encoding. Currently, this mapping of regions onto the image grid is stored explicitly and without any compression. One of the open problems of the SCIS algorithm is to incorporate a method to also store this mapping efficiently in terms of storage costs.

In Section 5.4 we shortly outlined our observations about the relationships of the distribution of the individual topological classes, total number of regions in an image, and the "amount" of texture in an image. Further research in this direction may result in a way to quantify the "amount" of texture in an image, which may provide a better understanding of current textural classifiers.

The SCIS algorithm was originally developed and proposed for 2-dimensional images,

however it can be extended to higher dimensions, so that it is for example able to segment whole sequences of images in a video. For this purpose it would be necessary to extend the topological classification using 3-dimensional Local Binary Patterns, and the identification of redundant edges. This identification and subsequent removal of redundant edges may provide, together with the minimization process for 3-dimensional combinatorial maps proposed in [Ill06], an even higher reduction in terms of the amount of edges.

CHAPTER 7

Conclusion

In this work we presented a new image segmentation algorithm which is based on Local Binary Patterns and the Combinatorial Pyramid, called Structurally Correct Image Segmentation (SCIS) algorithm. The main idea is to remove redundant edges, while at the same time preserving all reachability relationships and local structural correctness to perform a segmentation. We evaluated and compared the SCIS algorithm on the test images of the Berkeley Image Segmentation Dataset to five other algorithms using the Global Consistency Error (GCE) and Probability Rand Index (PRI) error metrics. A comparison of the SCIS algorithm to its simplified version, which does not take structural correctness into consideration, was performed and it was shown on both metrics that better results were achieved when taking structural constraints into consideration. The SCIS algorithm has shown comparably good results in the GCE metric to four existing algorithms, which are based on internal- and external contrast relationships, Minimum Spanning Trees, Mean-Shift clustering, and superpixel approaches. This performance is given due to the sequence at which different types of regions are merged, namely weakly textured and blurred regions are merged first, while highly textured regions are preserved. This segmentation behavior causes that the perceptually important content of the image is well preserved at a low number of regions, which can find an application in image compression. The PRI metric shows that the SCIS algorithm has the highest overall score when compared to the other five segmentation algorithms, and it also confirms that a refinement of segmentations takes place at regions where there is evidence of multiple levels of granularity of segmentations performed by human subjects.

APPENDIX A

Appendix

In this appendix we display plots with the exact segmentation results for all evaluated algorithms and the corresponding GCE- and PRI error curves. All the metric- (GCE, PRI), image type- (grayscale, color), and algorithm information corresponding to each plot can be found in the description below each one of them.

A.1 GCE Error Curves for Grayscale Images

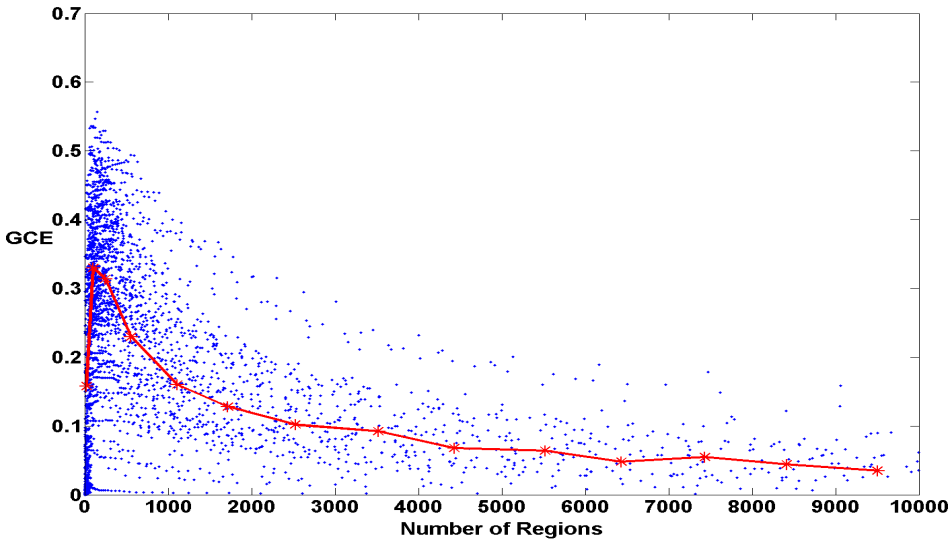


Figure A.1: GCE error curve of the *Greedy* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

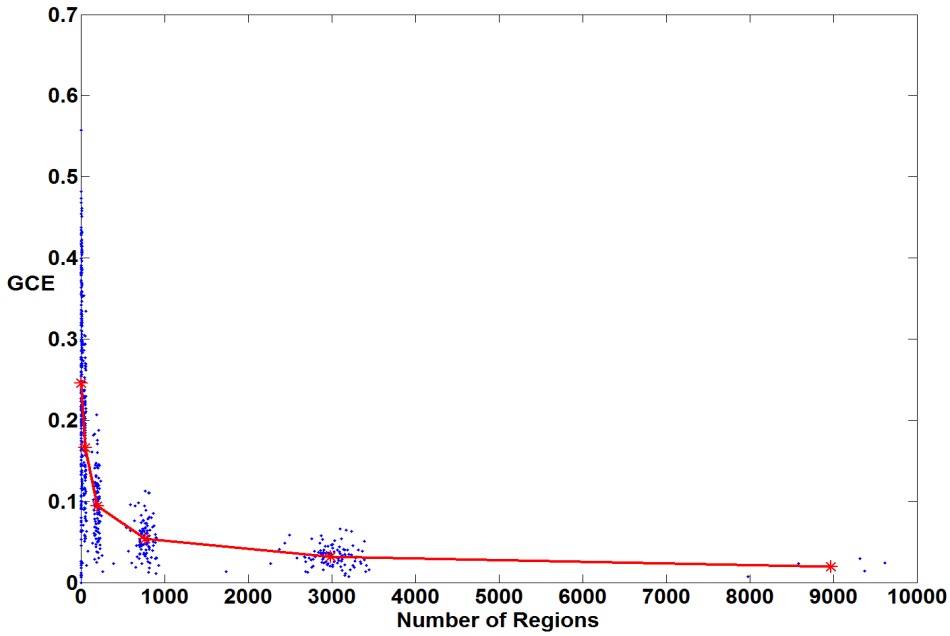


Figure A.2: GCE error curve of the *IntExtMST* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

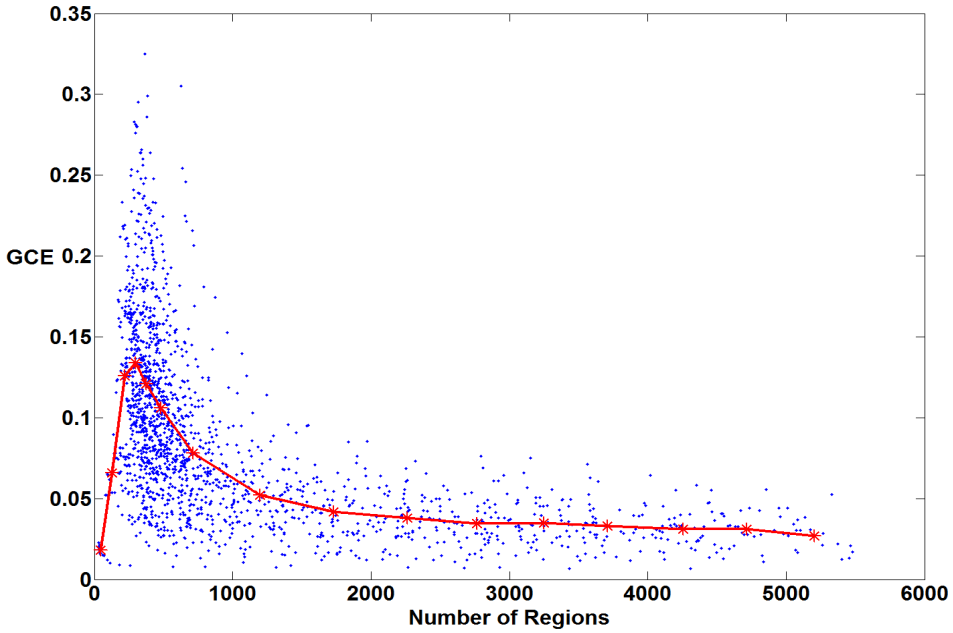


Figure A.3: GCE error curve of the *Eff-Graph* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

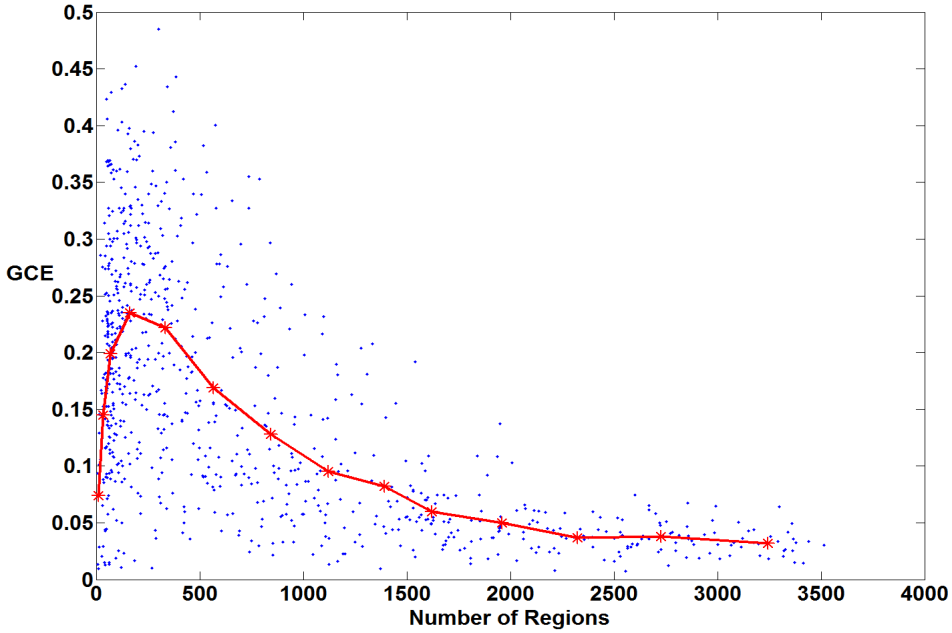


Figure A.4: GCE error curve of the *Mean-Shift* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

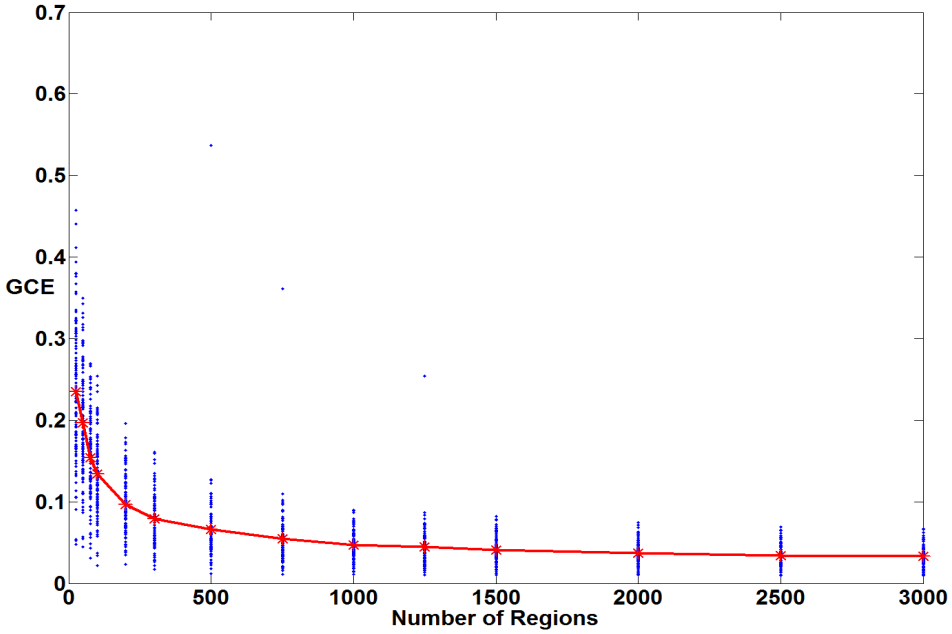


Figure A.5: GCE error curve of the *Turbopixels* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

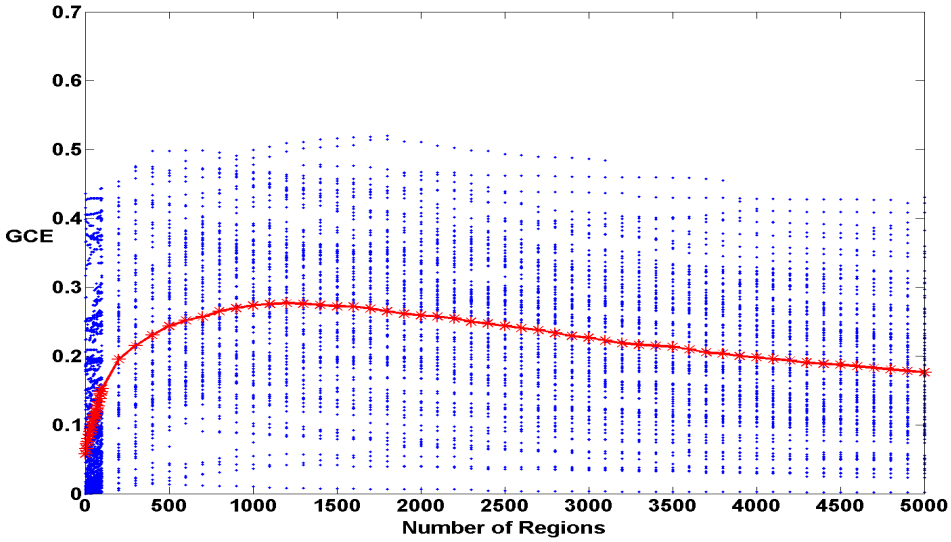


Figure A.6: GCE error curve of the *SCIS* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

A.2 PRI Error Curves for Grayscale Images

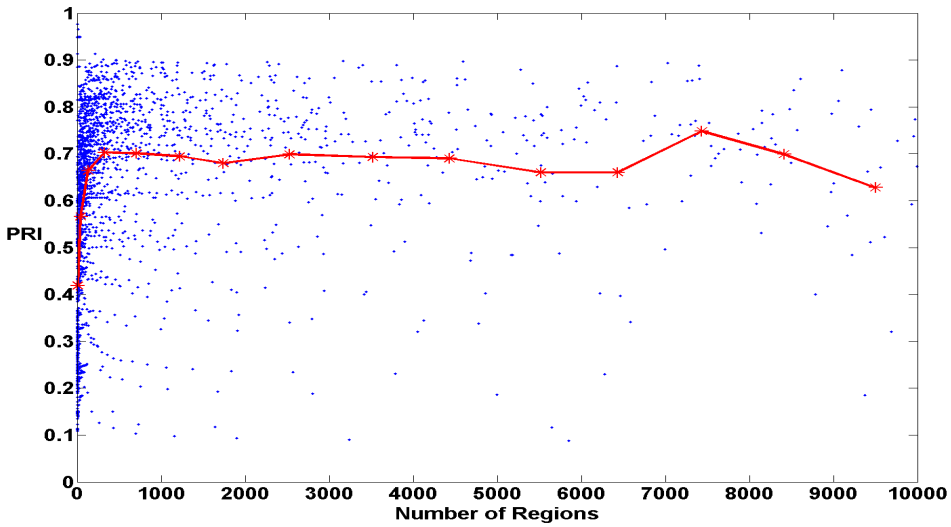


Figure A.7: PRI error curve of the *Greedy* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

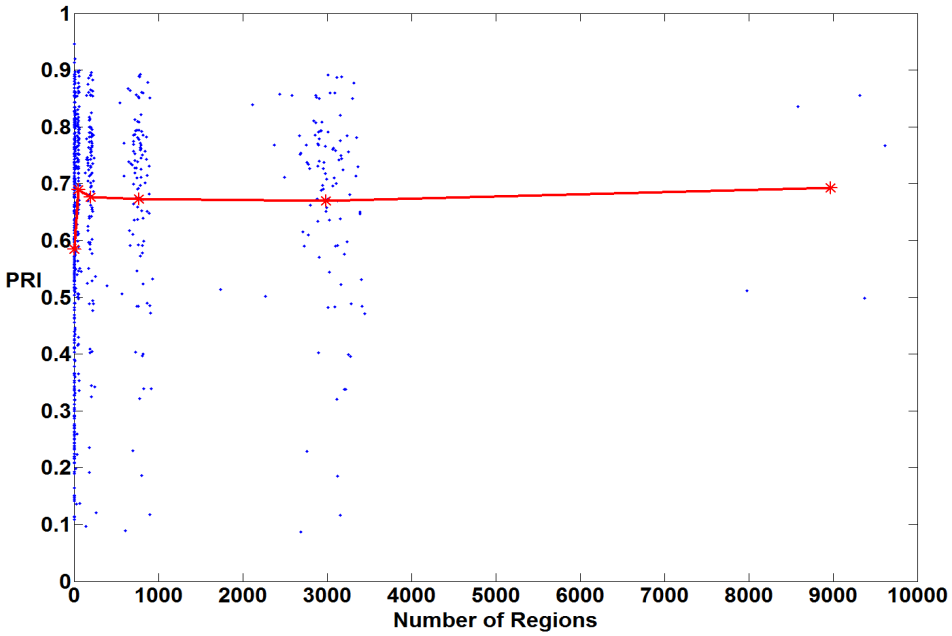


Figure A.8: PRI error curve of the *IntExtMST* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

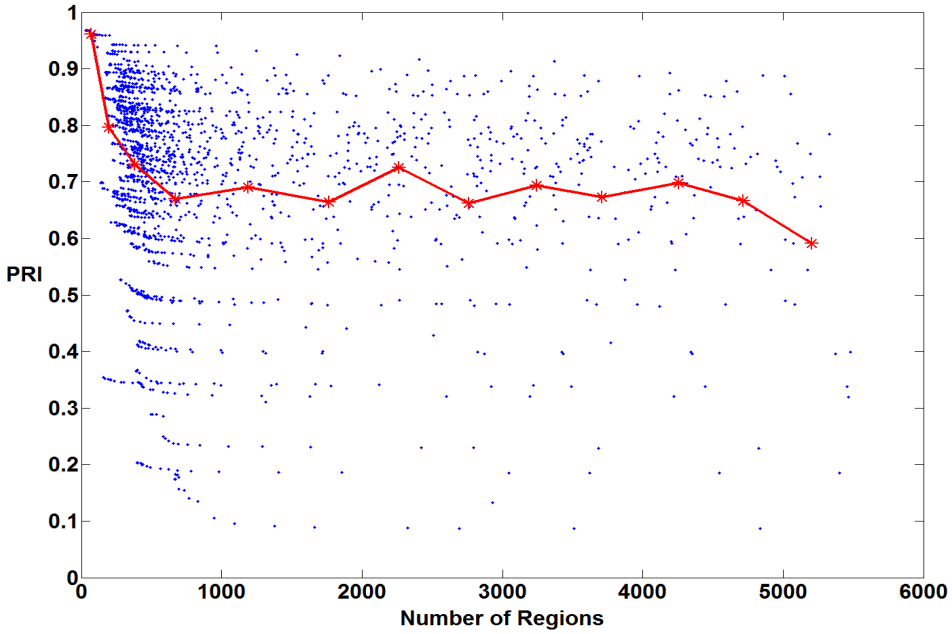


Figure A.9: PRI error curve of the *Eff-Graph* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

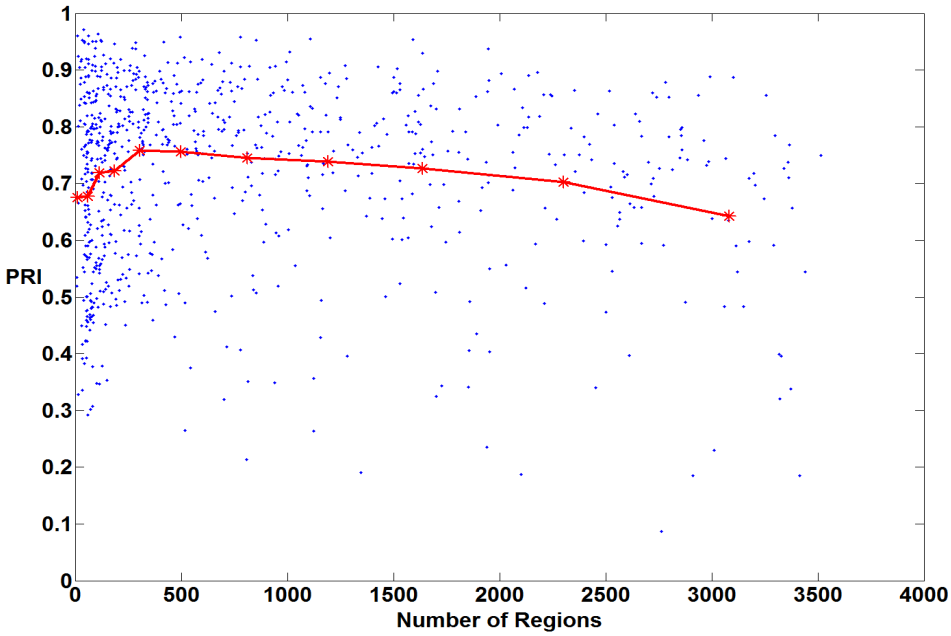


Figure A.10: PRI error curve of the *Mean-Shift* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

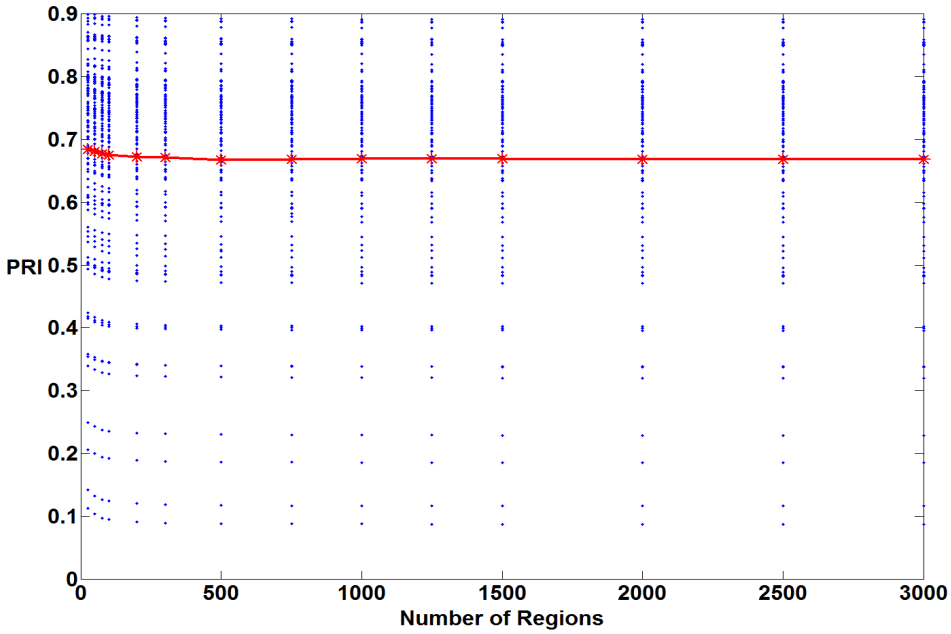


Figure A.11: PRI error curve of the *Turbopixels* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

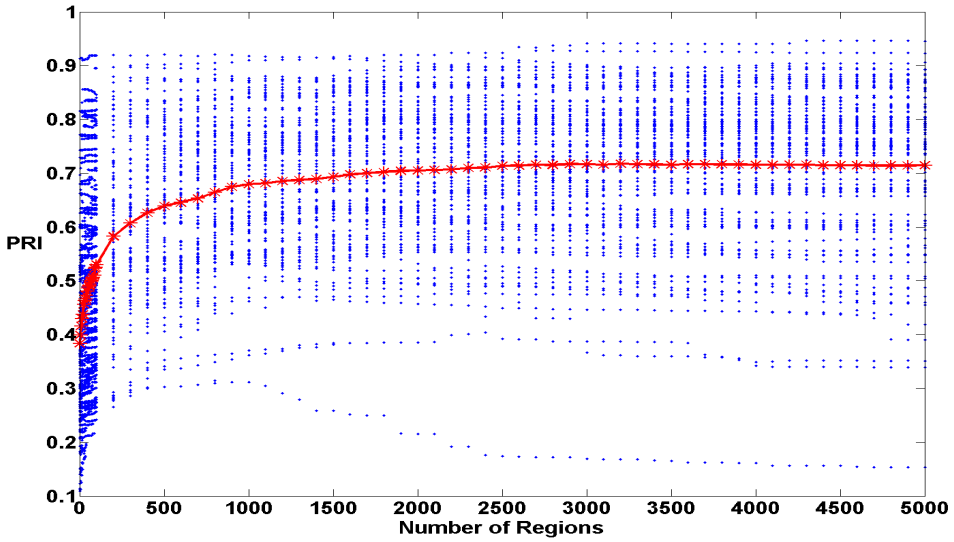


Figure A.12: PRI error curve of the *SCIS* algorithm for grayscale images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the PRI error curve is marked with red color.

A.3 GCE Error Curves for Color Images

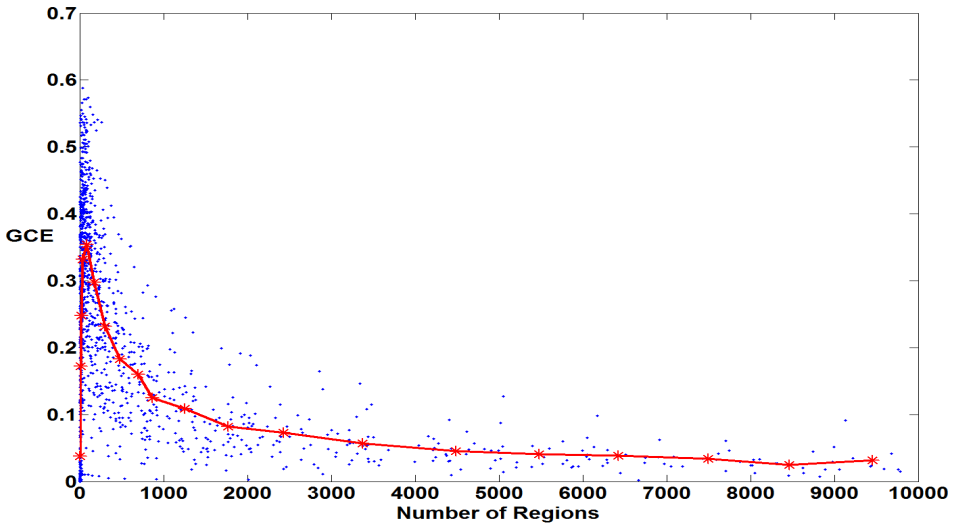


Figure A.13: GCE error curve of the *Greedy* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

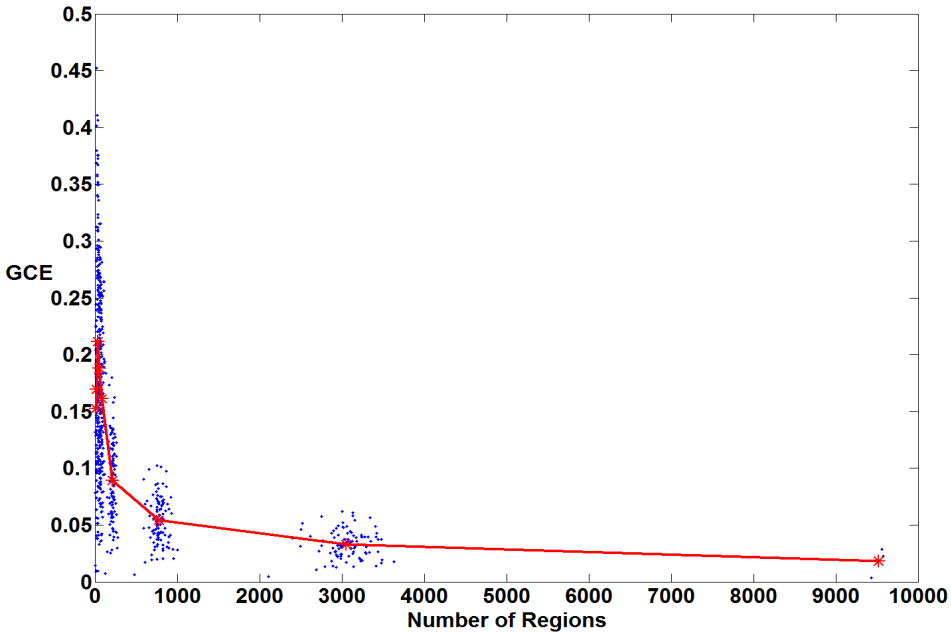


Figure A.14: GCE error curve of the *IntExtMST* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

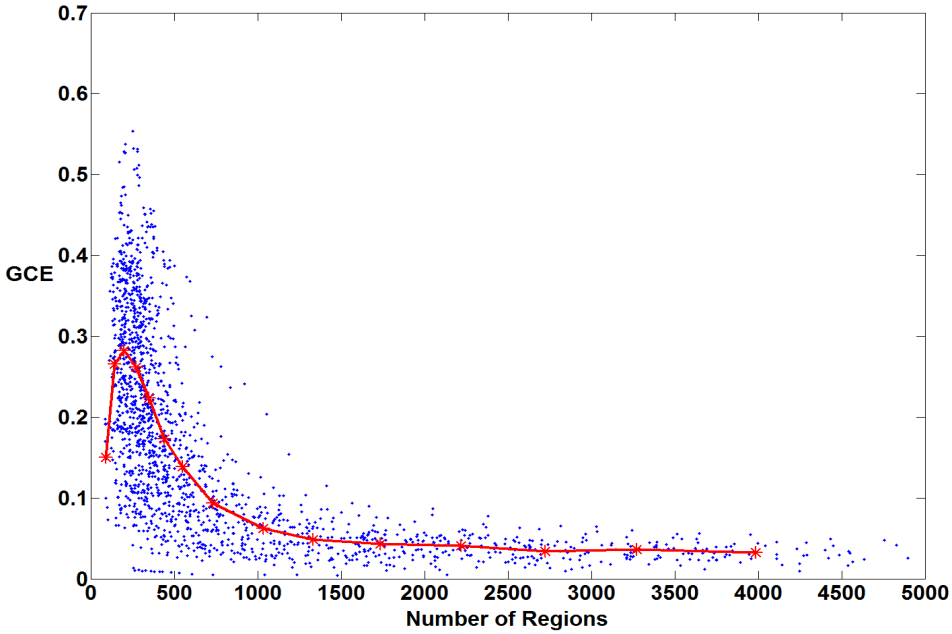


Figure A.15: GCE error curve of the *Eff-Graph* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

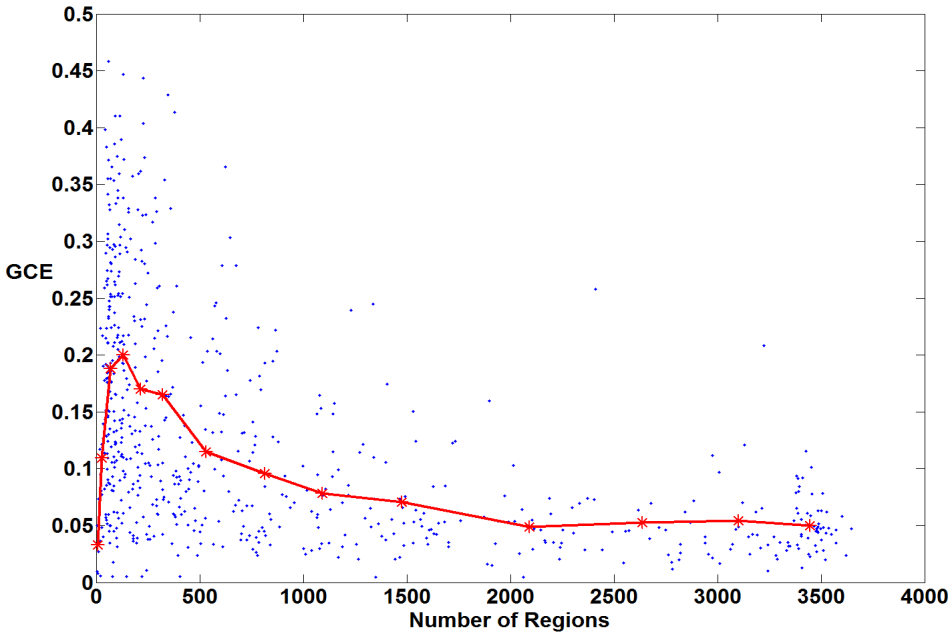


Figure A.16: GCE error curve of the *Mean-Shift* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

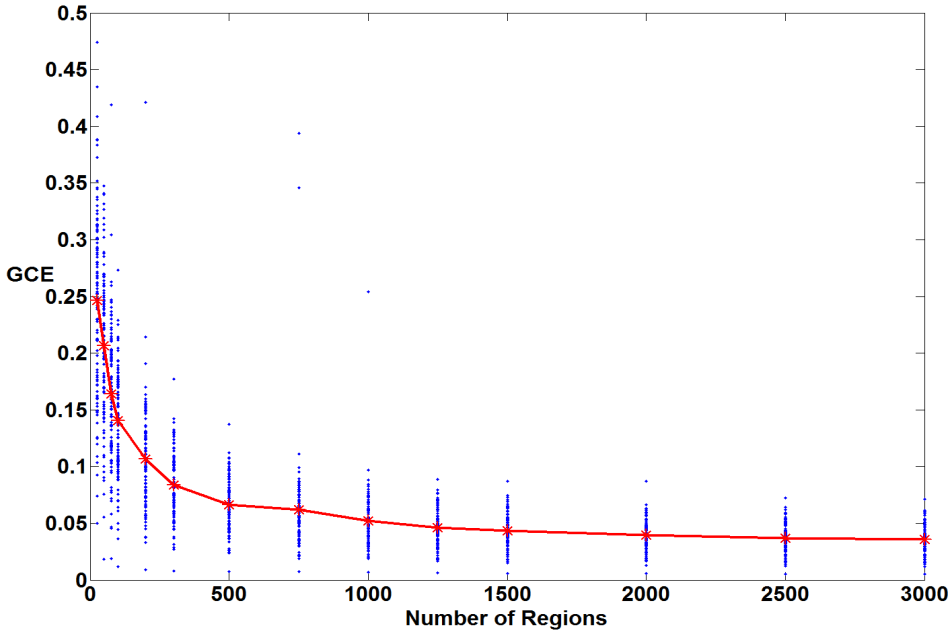


Figure A.17: GCE error curve of the *Turbopixels* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

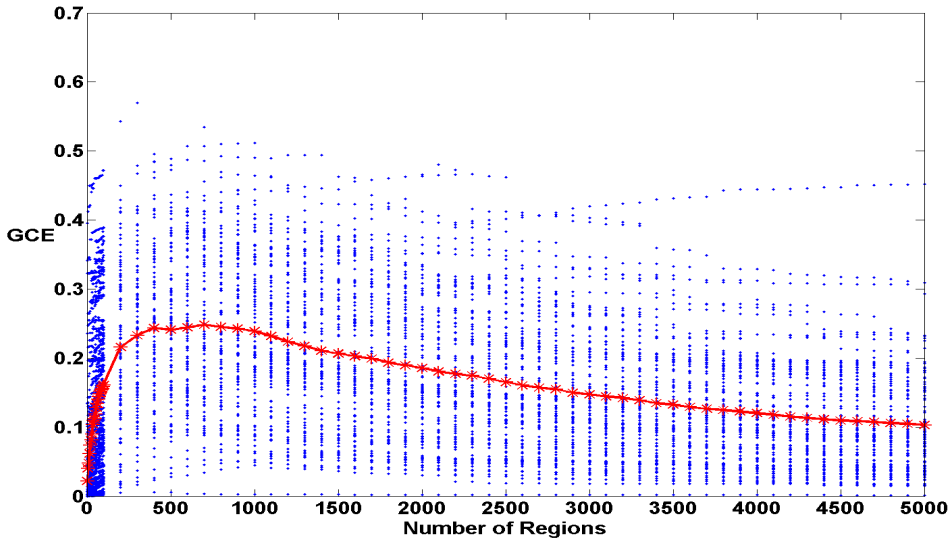


Figure A.18: GCE error curve of the *SCIS* algorithm for color images. The segmentation results are marked as blue dots, the mean GCE error for each bin is marked as a red star, and the GCE error curve is marked with red color.

A.4 PRI Error Curves for Color Images

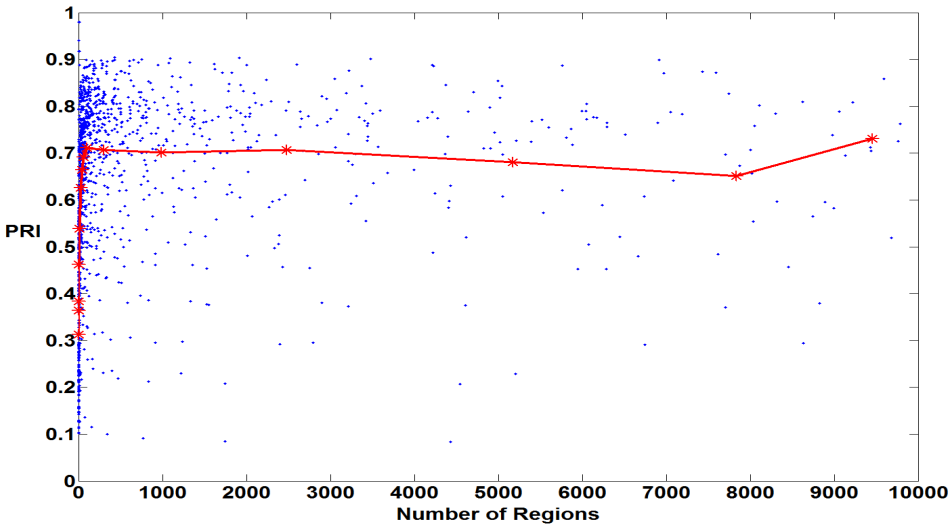


Figure A.19: PRI error curve of the *Greedy* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

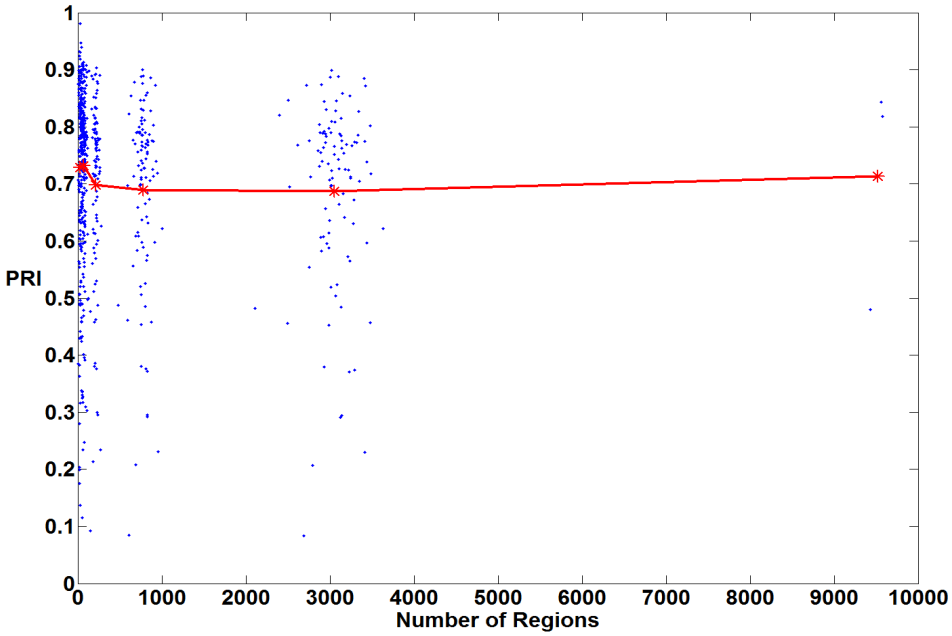


Figure A.20: PRI error curve of the *IntExtMST* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

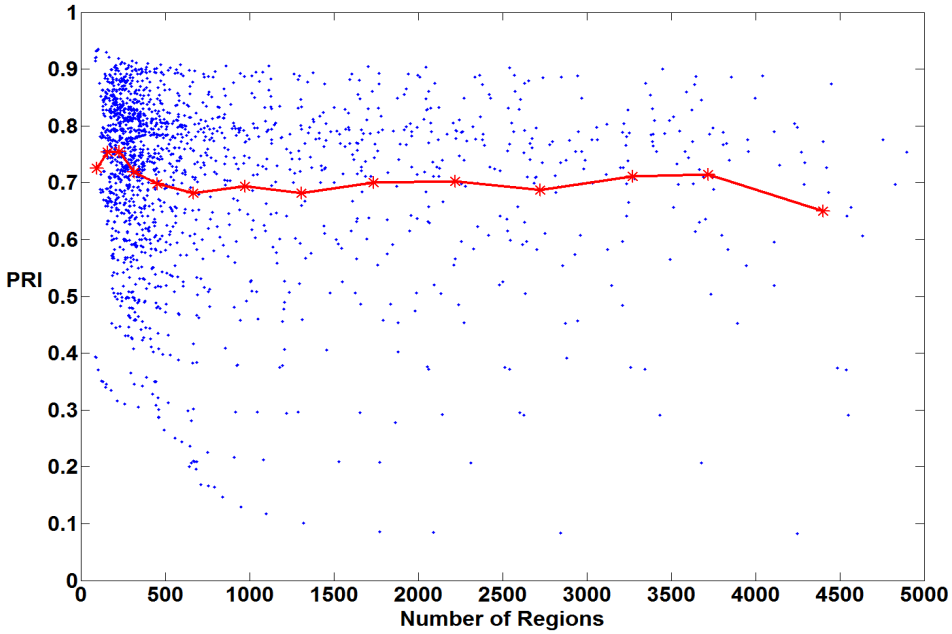


Figure A.21: PRI error curve of the *Eff-Graph* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

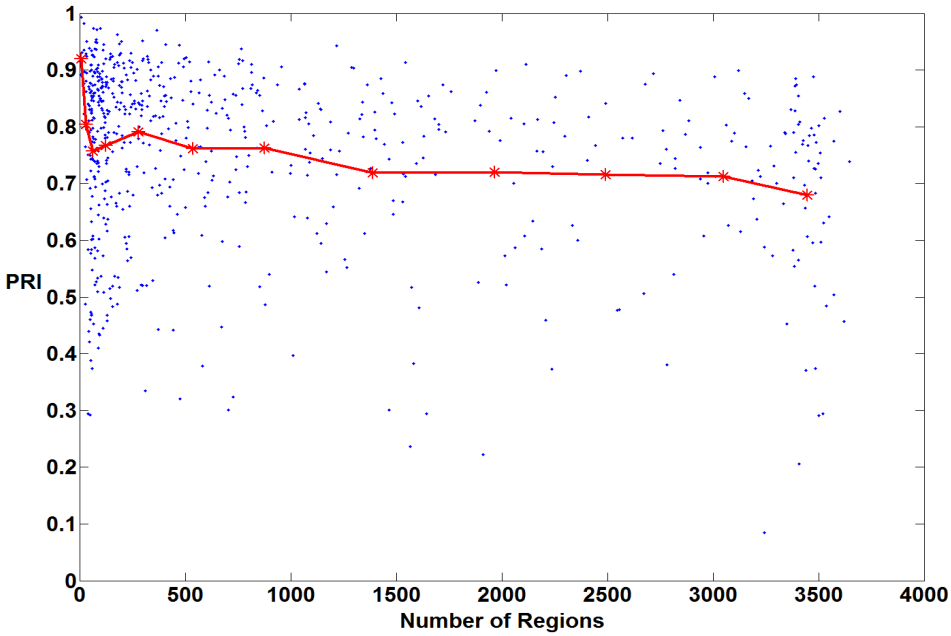


Figure A.22: PRI error curve of the *Mean-Shift* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

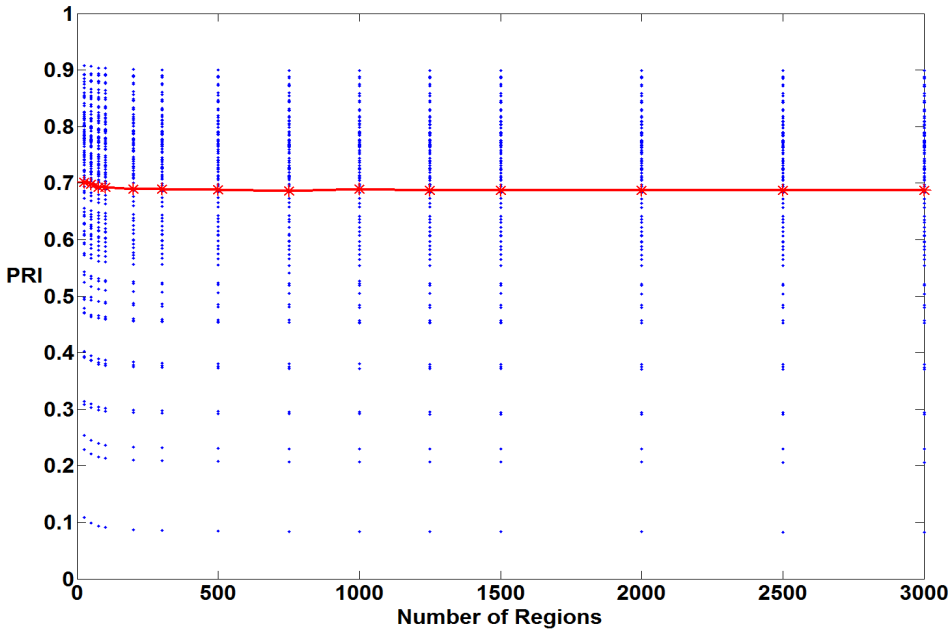


Figure A.23: PRI error curve of the *Turbopixels* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

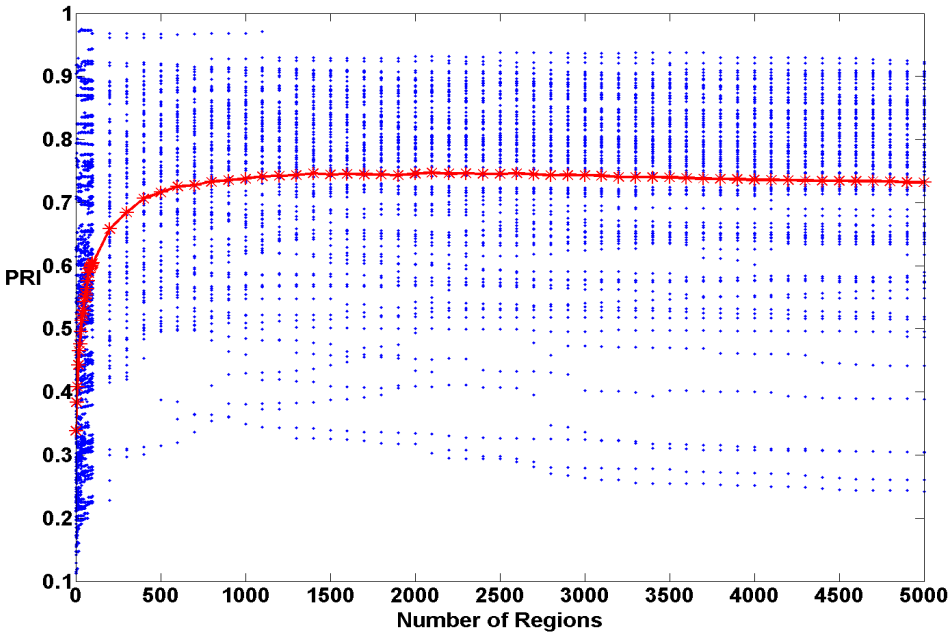


Figure A.24: PRI error curve of the *SCIS* algorithm for color images. The segmentation results are marked as blue dots, the mean PRI error for each bin is marked as a red star, and the GCE error curve is marked with red color.

Bibliography

[AAB+84] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid Methods in Image Processing. *RCA Engineer*, 29(6):33–41, 1984.

[AGU72] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM J. Comput.*, 1(2):131–137, 1972.

[BHR81] Peter J. Burt, Tsai H. Hong, and Azriel Rosenfeld. Segmentation and Estimation of Image Region Properties through Cooperative Hierarchial Computation. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(12):802–809, 1981.

[BK99] Luc Brun and Walter Kropatsch. Dual Contraction of Combinatorial Maps. Technical Report PRIP-TR-054, PRIP, TU Wien, 1999.

[BK01] Luc Brun and Walter G. Kropatsch. Introduction to Combinatorial Pyramids. In *Digital and Image Geometry*, volume 2243 of *Lecture Notes in Computer Science*, pages 108–128. Springer, 2001.

[BK03a] Luc Brun and Walter Kropatsch. Combinatorial pyramids. In Suvisoft, editor, *IEEE International conference on Image Processing (ICIP)*, volume II, pages 33–37, Barcelona, September 2003. IEEE.

[BK03b] Luc Brun and Walter Kropatsch. Construction of Combinatorial Pyramids. In Edwin Hancock and Mario Vento, editors, *Graph based Representations in Pattern Recognition*, volume 2726 of *LNCS*, pages 1–12, York, UK, June 2003. IAPR-TC15.

[Bor26] Otakar Borůvka. O jistém problému minimálnim. *Práce Mor. Přírodověd. Spol. v Brně (Acta Societ. Scienc. Natur. Moravicae)*, 3:37–58, 1926.

[Bru02] Luc Brun. *Traitement d'images couleur et Pyramides combinatoires*. Habilitation à diriger des recherches, Université de Reims, 2002.

[CGDK15] Martin Cerman, Rocio Gonzalez-Diaz, and Walter G. Kropatsch. LBP and Irregular Graph Pyramids. In *Computer Analysis of Images and Patterns*, volume 9257 of *Lecture Notes in Computer Science*. Springer International Publishing, 2015.

[CM02] Dorin Comaniciu and Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24:603–619, 2002.

[CSH+10] Jie Chen, Shiguang Shan, Chu He, Guoying Zhao, Matti Pietikäinen, Xilin Chen, and Wen Gao. WLD: A Robust Local Image Descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9):1705–1720, 2010.

[CZP08] Jie Chen, Guoying Zhao, and Matti Pietikäinen. Unsupervised Dynamic Texture Segmentation using Local Spatiotemporal Descriptors. In *International Conference on Pattern Recognition (ICPR)*, pages 1–4, 2008.

[CZP09] Jie Chen, Guoying Zhao, and Matti Pietikäinen. An Improved Local Descriptor and Threshold Learning for Unsupervised Dynamic Texture Segmentation. In *Proceedings of 12th International Conference on Computer Vision Workshops*, pages 460–467. IEEE Computer Society, 2009.

[DL14] Guillaume Damiand and Pascal Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing*. A K Peters/CRC Press, September 2014.

[FB10] S'ebastien Fourey and Luc Brun. Efficient encoding of n-D combinatorial pyramids. In *Proceedings of the International Conference on Pattern Recognition (ICPR'2010)*, Istanbul, Turkey, August 2010.

[FH04] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal on Computer Vision (IJCV)*, 59(2):167–181, September 2004.

[FMR+02] J. Freixenet, X. Muñoz, D. Raba, J. Martí, and X. Cufí. Yet Another Survey on Image Segmentation: Region and Boundary Information Integration. In *European Conference on Computer Vision (ECCV)*, volume 2352 of *Lecture Notes in Computer Science*, pages 408–422, 2002.

[GDKCL14] Rocio Gonzalez-Diaz, Walter G. Kropatsch, Martin Cerman, and Javier Lamar. Characterizing Configurations of Critical Points through LBP. In *International Workshop on Computational Topology in Image Context (CTIC) 2014*, September 2014. Extended Abstract.

[GHK11] Michael Gerstmayer, Yll Haxhimusa, and Walter G. Kropatsch. Hierarchical Interactive Image Segmentation Using Irregular Pyramids. In *Graph-based Representations in Pattern Recognition*, volume 6658 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 2011.

[GSDL06] Carine Grasset-Simon, Guillaume Damiand, and Pascal Lienhardt. *n*D generalized map pyramids: Definition, representations and basic operations. *Pattern Recognition*, 39(4):527 – 538, 2006. Graph-based Representations.

[Har69] Frank Harary. *Graph Theory.* Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

[Hax07] Yll Haxhimusa. *The Structurally Optimal Dual Graph Pyramid and its Application in Image Partitioning.* IOS Press, first edition, 2007.

[HGS+02] Yll Haxhimusa, Roland Glantz, Maamar Saib, Georg Langs, and Walter G. Kropatsch. Logarithmic Tapering Graph Pyramid. In *Pattern Recognition*, volume 2449 of *Lecture Notes in Computer Science*, pages 117–124. Springer Berlin Heidelberg, 2002.

[HK03] Yll Haxhimusa and Walter G. Kropatsch. Image Partitioning with Graph Pyramids. In C.Beleznai and Th. Schoegl, editors, *Proceedings of 27th Austrian Association of Pattern Recognition Workshop, OEAGM 2003*, pages 81–88, Laxenburg, Austria, 2003. Austrian Computer Society.

[HK04] Yll Haxhimusa and Walter G. Kropatsch. Segmentation Graph Hierarchies. In *Structural, Syntactic, and Statistical Pattern Recognition*, volume 3138 of *Lecture Notes in Computer Science*, pages 343–351. Springer Berlin Heidelberg, 2004.

[HP06] Marko Heikkilä and Matti Pietikäinen. A Texture-Based Method for Modeling the Background and Detecting Moving Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28:657–662, 2006.

[HPS06] Marko Heikkilä, Matti Pietikäinen, and Cordelia Schmid. Description of Interest Regions with Center-Symmetric Local Binary Patterns. In *Proceedings on The Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*, volume 4338 of *Lecture Notes in Computer Science*, pages 58–69. Springer Berlin Heidelberg, 2006.

[HS08] Allan Hanbury and Julian Stöttinger. On Segmentation Evaluation Metrics and Region Counts. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR)*, 2008.

[HSD73] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 3(6):610–621, November 1973.

[HW90] Dong-Che He and Li Wang. Texture Unit, Texture Spectrum, and Texture Analysis. In *IEEE Transactions on Geoscience and Remote Sensing*, volume 28, pages 509–512, 1990.

[IIKG08] Mabel Iglesias, Adrian Ion, Walter G. Kropatsch, and Edel García. Delineating Homology Generators in Graph Pyramids. In José Ruiz-Shulcloper and Walter G. Kropatsch, editors, *CIARP*, volume 5197 of *Lecture Notes in Computer Science*, pages 576–584. Springer, 2008.

[Ill06] Thomas Illetschko. Minimal Combinatorial Maps for Analyzing 3D Data. Technical Report PRIP-TR-110, PRIP, TU Wien, 2006.

[KBL07] Abraham Kandel, Horst Bunke, and Mark Last. *Applied Graph Theory in Computer Vision and Pattern Recognition.* Studies in Computational Intelligence. Springer, 2007.

[KHI07] Walter G. Kropatsch, Yll Haxhimusa, and Adrian Ion. Multiresolution Image Segmentations in Graph Pyramids. In Abraham Kandel, Horst Bunke, and Mark Last, editors, *Applied Graph Theory in Computer Vision and Pattern Recognition*, volume 52 of *Studies in Computational Intelligence*, pages 3–41. Springer Berlin Heidelberg, 2007.

[Kro98] Walter G. Kropatsch. From Equivalent Weighting Functions To Equivalent Contraction Kernels. In Emanuel Wenger and Leonid I. Dimitrov, editors, *Digital Image Processing and Computer Graphics (DIP-97): Applications in Humanities and Natural Sciences*, volume 3346, pages 310–320, 1998.

[Low99] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157. IEEE Computer Society, 1999.

[LSK+09] Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson, and Kaleem Siddiqi. TurboPixels: Fast Superpixels Using Geometric Flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 31(12):2290–2297, December 2009.

[LT03] Poh-Kok Loo and Chew-Lim Tan. Using Irregular Pyramid for Text Segmentation and Binarization of Gray Scale Image. In *Seventh International Conference on Document Analysis an Recognition*, pages 594–598. IEEE Computer Society, 2003.

[MFM04a] David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(5):530–549, May 2004.

[MFM04b] R. David Martin, C. Charless Fowlkes, and Jitendra Malik. Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(5):530–549, 2004.

[MFTM01] David R. Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *Proceedings of the 8th International Conference on Computer Vision (ICCV)*, volume 2, pages 416–423, July 2001.

[MMR91] Annick Montanvert, Peter Meer, and Azriel Rosenfeld. Hierarchical Image Analysis Using Irregular Tessellations. *IEEE Transactions on Pattern Recognition and Machine Learning*, 13(4):307–316, 1991.

[MP04] Topi Mäenpää and Matti Pietikäinen. Classification with Color and Texture: Jointly or Separately? *Pattern Recognition*, 37(8):1629–1640, 2004.

[MR06] Rebeca Marfil Robles. *Tracking Objects with the Bounded Irregular Pyramid*. PhD thesis, University of Malaga, 2006.

[MT69] Dennis M. Moyles and Gerald L. Thompson. An Algorithm for Finding a Minimum Equivalent Graph of a Digraph. *Journal of the ACM*, 16(3):455–460, July 1969.

[OP99] Timo Ojala and Matti Pietikäinen. Unsupervised Texture Segmentation using Feature Distributions. In *Pattern Recognition*, volume 32, pages 477–486, 1999.

[OPH96] Timo Ojala, Matti Pietikäinen, and David Harwood. A Comparative Study of Texture Measures with Classification based on Featured Distributions. *Pattern Recognition*, 29(1):51–59, January 1996.

[OPM02] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(7):971–987, 2002.

[PH15] Saúl Eduardo Pomares Hernández. The Minimal Dependency Relation for Causal Event Ordering in Distributed Computing. *Applied Mathematics & Information Sciences*, 9(1):pp.57–61, January 2015.

[PHZA11] Matti Pietikäinen, Abdenour Hadid, Guoying Zhao, and Timo Ahonen. Local Binary Patterns for Still Images. In *Computer Vision Using Local Binary Patterns*, pages 13–47. Springer, 2011.

[PZHA11] Matti Pietikäinen, Guoying Zhao, Abdenour Hadid, and Timo Ahonen. *Computer Vision Using Local Binary Patterns*. Number 40 in Computational Imaging and Vision. Springer, 2011.

[PZZ13] Bo Peng, Lei Zhang, and David Zhang. A Survey of Graph Theoretical Approaches to Image Segmentation. *Pattern Recognition*, 46(3):1020–1038, 2013.

[Ran71] William M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[Ros84] Azriel Rosenfeld. *Multiresolution Image Processing and Analysis*, volume 12 of *Springer Series in Information Sciences*. Springer, 1984.

[RRDR09] Shital Raut, M. Raghuvanshi, R. Dharaskar, and Adarsh Raut. Image Segmentation - A State-Of-Art Survey for Prediction. In *Proceedings of the 2009 International Conference on Advanced Computer Control (ICACC)*, pages 420–424. IEEE Computer Society, 2009.

[SHB93] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision*. Chapman & Hall, 1993.

[SM00] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, August 2000.

[SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 Color-Difference Formula: Implementation Notes, Supplementary Test Data, and Mathematical Observations. *Color Research & Application*, 30(1):21–30, 2005.

[TK14] Fuensanta Torres and Walter G. Kropatsch. Canonical Encoding of the Combinatorial Pyramid. In Zuzana Kúkelová and Jan Heller, editors, *Proceedings of the 19th Computer Vision Winter Workshop*, pages 118–125, February 2014.

[TP06] Markus Turtinen and Matti Pietikäinen. Contextual Analysis of Textured Scene Images. In *Proceedings on The 17th British Machine Vision Conference (BMVC)*, 2, pages 849–858. Edinburgh, UK, 2006.

[UH05] Ranjith Unnikrishnan and Martial Hebert. Measures of Similarity. *IEEE Workshop on Applications aof Computer Vision*, pages 394–400, 2005.

[UPH07] Ranjith Unnikrishnan, Caroline Pantofaru, and Martial Hebert. Toward Objective Evaluation of Image Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(6):929–944, June 2007.

[Wal14] Ben Waller. *On the Analysis of Structure in Texture*. PhD thesis, University of Southampton, March 2014.

[WDR76] Joan S. Weszka, Charles R. Dyer, and Azriel Rosenfeld. A Comparative Study of Texture Measures for Terrain Classification. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 6(4):269–285, 1976.

[Wil12] Virginia Vassilevska Williams. Multiplying Matrices Faster Than Coppersmith-Winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898. ACM, 2012.

[Zha96] Yu-Jin Zhang. A Survey on Evaluation Methods for Image Segmentation. *Pattern Recognition*, 29(8):1335–1346, 1996.