



國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering
College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

運用超像素的即時互動式影像切割

Real-Time Interactive Image Segmentation with Superpixel
Pre-segmentation

林家蓉

Chia-Jung Lin

指導教授：丁建均 博士

Advisor: Jian-Jiun Ding, Ph.D.

中華民國 103 年 7 月

July, 2014

誌謝



首先感謝我的父母，一直以來包容我的幼稚及任性，並持續支持我走我選擇的路，即使我仍並不十分確定自己想要的是什麼。感謝我的弟妹時常幫我分憂，未來幾年分別的日子我會很想念很想念你們的。

碩一上忙著實習，碩一下出國交換，這一年沒有和實驗室的同學和學長姐們有太多交流，十分遺憾。感謝實驗室的同屆同學，盈柔、思維、胡豪，感謝有你們一起修課一同耍廢抱怨和表人。感謝柏宏和信慧，一直以來你們用很不同的風格帶領著這個實驗室。感謝學弟妹們，浩軒、士昌、景文、麒偉、奕帆、品萱，這一年與你們相處愉快，祝福你們未來一年也能在彼此扶持之下順利度過。

感謝哲勳將近四年的陪伴，是你讓我成為了現在的我。感謝為碩，我很懷念與你談心的時光。感謝李鎬，你總是不談心，但是在課業上給了我許多幫助，也很大程度地影響了我大三之後走的方向。感謝圈圈陪我走過碩二的這段時光，認識你多年才突然覺得我們如此相似，祝我們都幸福。感謝逸祥這麼好約，每次都分享身旁友人幼稚的舉動，讓我覺得自己十分成熟(?)順便暫時忘記煩憂。

謝謝威昇，你是我唯一不後悔這兩年的原因，你讓我看見不曾奢望過的美好風景。希望此去經年，仍是我陪著你追夢。

感謝其他以上未曾提到，但曾影響我或幫助過我的人。抱歉環境破壞已經很嚴重了我還浪費這麼多紙(哭)，整本最有價值的就是這一頁，看完請直接闔起來，謝謝大家。

中文摘要



影像切割一直是計算機視覺中一個重要的研究課題。這是因為電腦很難完全自動地從背景分割出目標，這也導致了互動式影像分割研究的興起。互動式影像分割藉由使用者的引導及修正來切割出理想的結果。然而，雖然稱為“互動”式影像切割，近期的許多研究仍未做到能即時回饋。

在本篇論文中，我們著重於縮短切割系統的反應時間。為了產生即時的結果，我們提出充分利用使用者仍在與系統互動的時間(例如正在標記前景或背景的大概位置)，我們將演算法分為兩部份，其中只有第二部分會造成使用者可能察覺到的系統延遲。我們也提出採用的紋理特徵以降低切割的錯誤率，尤其當前景和背景具有相似的顏色的時候。實驗結果表明，在人眼的反應系統下，我們的演算法和框架可以達到視為“即時”的反應時間，這樣的框架同時也可以被其他演算法拿來應用。

關鍵字：互動式影像切割、超像素、像切割、即時系統

ABSTRACT



Image segmentation has been a major research topic in computer vision. It is hard to segment the object from the background fully automatically, and solving the problem leads to the researches in interactive image segmentation, which incorporates user intervention to guide the segmentation process and to correct anomalies in the segmentation results. Although called “interactive,” real-time response is not yet reachable by many recent methods.

In this thesis, we focus on shorten the response time of the segmentation system. To produce real-time result, we propose making use of the time when user interacts with the system. We separate our algorithm into two parts, where the response time is only contributed by the later part. We adopt texture feature to enhance the results in images where foreground and background have similar colors. Experimental results show that our implementation achieves short response time that could be perceived as “instant.” The framework can also be applied by other different algorithms to shorten the response time.

Index terms: Interactive Image Segmentation, Superpixel, Graph Cut, Real-Time System

CONTENTS



口試委員會審定書	#
誌謝	i
中文摘要	ii
ABSTRACT	iii
CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	x
Chapter 1 Introduction.....	1
1.1 Motivation.....	1
1.2 Main Contribution	2
1.3 Organization	2
Chapter 2 Interactive Image Segmentation.....	3
2.1 Image Segmentation	3
2.2 Interactive Segmentation	5
2.3 Prior Works on Interactive Segmentation.....	6
2.3.1 Graph Cut	6
2.3.2 Random Walker.....	7
2.3.3 Shortest Path (Geodesic)	7
Chapter 3 Graph Cut Method	9
3.1 Preliminaries	9
3.2 Max-Flow/Min-Cut Theorem	10
3.3 Graph Cut.....	11

3.4	Graph Cut Segmentation.....	12
3.4.1	Basic Ideas and Background Information	12
3.4.2	Hard Constraints.....	16
3.5	Issues in Graph Cut Algorithm	19
3.5.1	The Shrinking Bias.....	19
3.5.2	Efficiency	21
3.5.3	Vulnerability to Noise	21
3.6	Prior Works	22
3.6.1	Speed-up based graph cut.....	22
3.6.2	Interactive-based graph cut	23
3.6.3	Shape prior-based graph cut.....	23
3.7	Lazy Snapping[5].....	25
Chapter 4	Overview of Superpixel Methods	29
4.1	Introduction.....	29
4.2	Gradient-Ascent-Based Algorithms.....	31
4.3	Graph-Based Algorithms	33
4.4	Remarks	34
4.5	Benchmark and Comparison.....	35
4.5.1	Benchmark	35
4.5.2	Comparison Results	37
Chapter 5	Proposed Interactive Image Segmentation Method.....	43
5.1	Observation.....	43
5.2	User Interface (UI) Design	44
5.3	Framework.....	45
5.4	Desired Properties of Superpixel Segmentation	46

5.5	Proposed Segmentation Algorithm	48
5.5.1	The Regional Term	48
5.5.2	The Boundary Term	50
5.5.3	Max-Flow/Min-Cut Algorithm	50
5.6	Summary	51
Chapter 6	Experimental Results and Discussion	53
6.1	Grabcut Benchmark	53
6.2	Discussion	65
6.2.1	Processing Time	65
6.2.2	Achievable Segmentation Error	70
6.2.3	Graph Cut Segmentation Issues Revisited	71
6.3	Summary	72
Chapter 7	Conclusion and Future Work	73
7.1	Conclusion	73
7.2	Future Work	74
Appendix A		
A.1	Simple Linear Iterative Clustering Superpixel (SLIC)	75
REFERENCE		81

LIST OF FIGURES

Fig. 2.1	Example of medical image segmentation. [1] (a) Magnetic resonance heart image; (b) image after segmented into three classes	4
Fig. 3.1	An example of a flow network.....	10
Fig. 3.2	Illustration of s-t graph. The image pixels correspond to the neighbor nodes in the graph(except s and t nodes). The orange lines in the graph are n-links and the black lines are t-links.	12
Fig. 3.3	Illustration of graph cut for image segmentation. The cut is corresponding to the minimal energy of (3.1)	15
Fig. 3.4	The optimal object segment (red tinted area in (c)) finds a balance between “region” and “boundary” terms in (2). The solution is computed using graph cuts as explained in Section 2.5. Some ruggedness of the segmentation boundary is due to metrication artifacts that can be realized by graph cuts in textureless regions.[3].....	17
Fig. 3.5	Illustration of shrinking bias. A_{12} and A_0 are some homogeneous areas. Graph-cut methods may short-cut across the desired object along B_1 instead of following the true edge B_2 because less cost is accrued traversing the shorter path.	21
Fig. 3.6	Lazy Snapping is an interactive image cutout system, consisting of two steps: a quick object marking step and a simple boundary editing step. In (b), only 2 (yellow) lines are drawn to indicate the foreground, and another (blue) line to indicate the background. All these lines are far away from the true object boundary. In (c), an accurate boundary can be obtained by simply clicking and dragging a few polygon vertices in the zoomed-in view. In (d), the	

	cutout is composed on another Van Gogh painting.	25
Fig. 3.7	(a) A small region by the pre-segmentation. (b) The nodes and edges for the graph cut algorithm with pre-segmentation. (c) The boundary output by the graph cut segmentation.	27
Fig. 4.1	An superpixel segmentation example. The input image (a) is segmented into 25 (middle part, top-left corner) and 250 (middle part, bottom-right) superpixels in (b). The segmentation result is shown with mean colors of each region in (c).....	29
Fig. 4.2	Visual comparison of superpixels produced by various methods. The average superpixel size in the upper left of each image is 100 pixels, and 300 in the lower right. Alternating rows show each segmented image followed by a detail of the center of each image. [48]	34
Fig. 4.3	Illustration of undersegmentation error. A ground truth segment (green) is covered by three superpixels (A,B,C), that can flood over the groundtruth segment border.[43]	36
Fig. 4.4	Runtime of segmentation algorithms (log-scale) [43].	40
Fig. 4.5	Boundary recall of segmentation algorithms on Neubert and Protzel's benchmark(top-left is better).[43].....	41
Fig. 4.6	Undersegmentation error of segmentation algorithms on Neubert & Protzel's benchmark(bottom-left is better). [43]	41
Fig. 4.7	Visual Comparison of superpixel segmentation results. The bumbers on the left side indicate the number of superpixels in one image.....	42
Fig. 5.1	Prototype of user interface.	44
Fig. 5.2	Flowchart of our system.	45
Fig. 6.1	Example of GrabCut dataset. Left column: original image. Middle column:	

the trimap image with user defined foreground (white) background (dark grey) and unknown (light grey). Right column: ground truth.	54
Fig. 6.2 Visual comparison on dataset [2].	59
Fig. 6.3 Visual comparison on dataset [2]. The error rate of (c)-(h) are 5.19%, 4.39%, 9.15%, 15.06%, 14.20% and 9.42% respectively.	60
Fig. 6.4 Visual comparison on dataset [2].	61
Fig. 6.5 Visual comparison on dataset [2].	62
Fig. 6.6 Visual comparison on dataset [2].	63
Fig. 6.7 Visual comparison on dataset [2].	64
Fig. 6.8 Response time with respect to different number of ERS superpixels.	66
Fig. 6.9 Processing time of computing energy term with respect to different number of ERS superpixels.	66
Fig. 6.10 Processing time of with respect to different number of ERS superpixels.	67
Fig. 6.11 Response time of 50 images with different node ratio. (ERS)	67
Fig. 6.12 Response time with respect to different number of SLICO superpixels.	68
Fig. 6.13 Processing time of computing energy term with respect to different number of SLICO superpixels.	68
Fig. 6.14 Processing time of with respect to different number of SLICO superpixels.	69
Fig. 6.15 Response time of 50 images with different node ratio. (SLICO).	69
Fig. 6.16 Visual comparison of our failure case.	71
Fig. A.1 Reducing the superpixel search regions. (a) standard k-means searches the entire image. (b) SLIC searches a limit region.	76
Fig. A.2 Segmentation results of SLIC and SLICO.	79

LIST OF TABLES

Table 3.1 Weight of the s-t graph. P represent all the pixels in the original image and

$$K = 1 + \max_{p \in P} \sum_{q: \{p,q\} \in N} B_{p,q} \dots\dots\dots 17$$

Table 3.2 Manually assigned regional values.18

Table 3.3 Definitions of $R_i(l_i)$ 26

Table 3.4 Core algorithm of first step in Lazy Snapping.....28

Table 4.1 Summary of existing superpixel algorithms.34

Table 4.2 Runtime (in seconds) of segmentation algorithms[43].....40

Table 5.1 Definitions of regional term.....49

Table 5.2 Algorithm of ur interactive segmentation framework51

Table 5.3 Algorithm of ur interactive segmentation framework51

Table 6.1 Performance evaluation Microsoft GrabCut Database [3].56

Table 6.2 Algorithms of Calculating ASE58

Table 6.3 Preprocessing time of two superpixel methods (in second).70

Table 6.4 Error rate of two superpixel methods and various node ratio.....70

Table 6.5 Error Rate of segmentation results on noisy dataset.....72

Table A.1 Algorithm of SLICO superpixel segmentation.80

Chapter 1 Introduction

1.1 Motivation



Photo editor is probably the most popular application on PCs or smart devices, since photo sharing has become such an important part of social life. At the same time, users have become stricter than ever with the response time of applications. Image segmentation is an important part of photo editing, because it is usually a pre-step for other local editing. We look through recent researches on interactive segmentation; almost all methods need users to do further correction. In terms of user behavior, we think giving timely response is more important than the accuracy. That means we are willing to sacrifice a few percent of accuracy for a real-time system without any delay. Since 4% of error rates needs further correction as much as 7 % of error rates. To note that the system we discuss here is more suitable for single image segmentation. Interactive video segmentation (segment through a series of continuous frames with only key frame marked by the user) demands higher accuracy and thus has different priority with us. Therefore, in this thesis, we focus on designing a real-time interactive segmentation system that generates result instantly while retain high accuracy.

1.2 Main Contribution

In this thesis, we review the researches in interactive image segmentation and superpixel segmentation. Based on superpixel segmentation and graph cut segmentation, we propose a novel framework that focuses on improving user experience by shortening the response time of the system. Experimental results show that the response time of our system can be perceived as “instant.” Additionally, we propose using texture information to retain the segmentation accuracy. Our results do not achieve the lowest error rate among recent methods, but are visually comparable to state-of-the-art methods in most of the test images.

1.3 Organization

This thesis is organized as follows: in Chapter 2, an overview of interactive image segmentation is presented. A detailed introduction of graph cut segmentation, including the original method and the extended ones in recent years, is given in Chapter 3. In Chapter 4, we give an overview of superpixel methods, both for conventional methods and modern approaches. In Chapter 5, we propose a novel framework, which enables the segmentation system to produce real-time result. Experimental results and discussion are given in Chapter 6. Finally, the conclusion and future work are given in Chapter 7.

Chapter 2 Interactive Image Segmentation

Interactive segmentation is one of the research subfields in image segmentation. In this chapter, we first introduce the concept and possible applications of image segmentation. Then, we explain the difficulties in image segmentation, which lead to the research in interactive image segmentation. Next, we cover the basic ideas of interactive segmentation. In the end, we briefly review three of the popular approaches to interactive image segmentation.

2.1 Image Segmentation

In computer vision, image segmentation is the process of partitioning an image into “meaningful” regions. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics. The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region is similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).

Image segmentation is a fundamental task in a large number of applications in computer vision. More generally, effective image segmentation is a key to better scene understanding and object recognition. For instance, in medical imaging, the resulting contours after image segmentation can be used to locate tumors, measure tissue volumes or do virtual surgery simulation. Fig. 2.1 shows an example of medical image segmentation. The image is roughly segmented into three regions by their difference in texture and intensity.

There are three main segmentation categories: fully automatic methods, semi-supervised (also called interactive or seeded) methods, and completely manual ones. The results produced by fully automatic methods are still far from matching human performance, especially for natural images. The difficulty of the task at hand and the limitations of the input data (real images differ from their models because of noise, occlusion, clutter, etc.) often lead to ill-posed problems. The segmentation process itself is in general ill-defined without additional prior knowledge: Homogeneity in some a priori feature space such as color or texture is not a sufficient criterion to define a meaningful, unambiguous image partition. People probably rely on a combination of low-level information (e.g., color, texture, contours) and high-level knowledge (e.g., shape priors, semantic cues) to resolve these ambiguities. Even with prior knowledge, automatic segmentation remains a challenging and active research area.

On the other hand, interactive segmentation allows some form of human intervention to guide the segmentation process and to correct anomalies in the segmentation results.

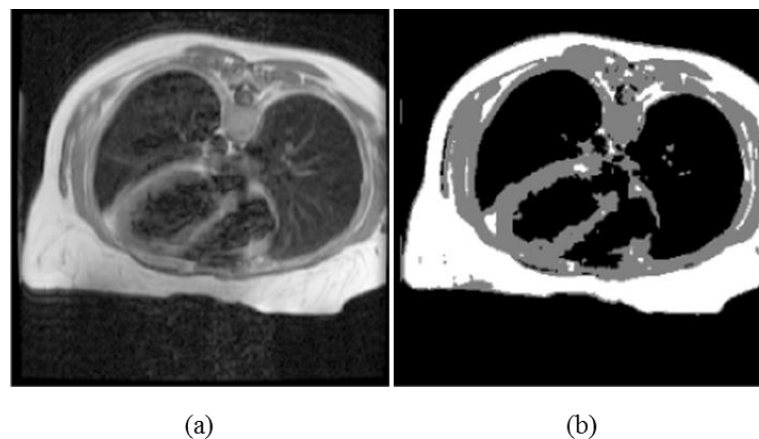


Fig. 2.1 Example of medical image segmentation. [1] (a) Magnetic resonance heart image; (b) image after segmented into three classes

2.2 Interactive Segmentation

A practical interactive segmentation algorithm usually contains these four qualities:

1. Fast computation
2. Fast editing
3. An ability to produce an arbitrary segmentation with enough interaction
4. Intuitive segmentations.

Many forms of interaction have been proposed, ranging from loosely tracing the desired boundary (e.g., [9],[10],[11],[13],[14]) to loosely marking parts of the desired object and/or background (e.g., [3],[4],[5],[21],[22],[23],[26]) to loosely placing a bounding box around the desired object (e.g., [2], [16]). In all forms, the goal is to allow the user to accurately select objects of interest with minimal effort.

We focus here on approaches where the user marks “scribbles” on parts of the desired foreground and background regions to seed the segmentation. Such approaches are popular because they generally require less precise input from the user, allowing them to loosely mark broader interior regions instead of more finely tracing near object boundaries, though each approach can sometimes be advantageous. Allowing the user to draw a bounding box [2] is simpler in many cases, though may not provide sufficient control in all cases, in which scribble-based corrections are often employed to refine the results.

Many methods for seeded segmentation expand outward from the seeds to selectively fill the desired region, either explicitly [22], [23], [24] or conceptually [21]. Because these approaches work from the interior of the selected object outwards and do not explicitly consider the object boundary, they are particularly useful for selecting objects with complex boundaries such as those with long, thin parts. However, because



these expansions are monotonic, these approaches suffer from a bias that favors shorter paths back to the seeds. As a result, they can be sensitive to seed placement, as illustrated for geodesic segmentation in Figure 2. Because they lack an explicit edge component, these methods may also fail to accurately localize object boundaries. The stronger the image edges are, the more likely these methods are to make these transitions here, but this is not guaranteed, as illustrated for geodesic segmentation in Figure 3.

2.3 Prior Works on Interactive Segmentation

The algorithms reviewed in this section view an image as a graph where each pixel corresponds to a node and the weighting of edges reflect changes in image intensity, color or other features.

2.3.1 Graph Cut

Our proposed algorithm is based on graph cut. Thus, we only give a brief introduction here, and we will give a more thorough review in Chapter 3. The labeling produced by the graph cut algorithm is determined by finding the minimum cut between the foreground and background seeds via a maximum flow computation. The original work on graph cut for interactive image segmentation was produced by Boykov and Jolly[3], and this work has been subsequently extended by several groups to employ different features [9] or user interfaces [2], [5]. Although graph cut is relatively new, the use of minimal surfaces in segmentation has been a common theme in computer vision for a long time [28], [27] and other boundary-based user interfaces have been previously employed [29], [30], [31].

2.3.2 Random Walker

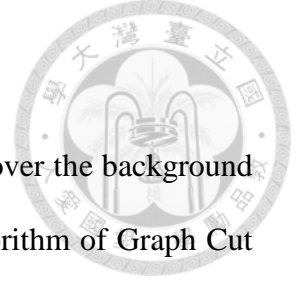
The random walker algorithm of Grady [22] is also formulated on a weighted graph and determines labels for the unseeded nodes by assigning the pixel to the seed for which it is most likely to send a random walker. This algorithm may also be interpreted as assigning the unlabeled pixels to the seeds for which there is a minimum diffusion distance [13], as a semi-supervised transduction learning algorithm [17] or as an interactive version of normalized cuts [33], [23]. Additionally, popular image matting algorithms based on quadratic minimization with the Laplacian matrix may be interpreted as employing the same approach for grouping pixels, albeit with different strategies to determine the edge weighting function [26]. Diffusion distances avoid segmentation leaking and the shrinking bias, but the segmentation boundary may be more strongly affected by seed location than with graph cuts [34].

2.3.3 Shortest Path (Geodesic)

The shortest path algorithm assigns each pixel to the foreground label if there is a shorter path from that pixel to a foreground seed than to any background seed, where paths are weighted by image content in the same manner as with the GC and RW approaches. This approach was recently popularized by Bai and Sapiro [5], but variants of this idea have appeared in other sources [16], [3], [18]. The primary advantage of this algorithm is speed and prevention of a shrinking bias. However, it exhibits stronger dependence on the seed locations than the Random Walker approach [34], is more likely to leak through weak boundaries (since a single good path is sufficient for connectivity) and exhibits metrickation artifacts on a 4-connected lattice.



Chapter 3 Graph Cut Method



In this chapter, we focus on Graph Cut approaches. First, we cover the background in graph theory in section 3.1-3.3. Then, we introduce the core algorithm of Graph Cut segmentation in section 3.4. In section 3.5, we discuss the weakness in this popular approach. Then, in section 3.6, we review the three main extensions based on Graph Cut segmentation. Finally, we give a thorough introduction to Lazy Snapping [5], which is one of the classic work aiming at improving processing time.

3.1 Preliminaries

- Graph representation: We use $G=(V,E)$ to denote an undirected graph where V is the vertex set and E is the edge set. The vertices and edges are denoted by v_i and $e_{i,j}$ respectively. The edge weights are denoted w_{ij} . In an undirected graph, the edge weights are symmetric, that is, $w_{ij} = w_{ji}$.
- Cut: In a graph $G=(V,E)$, a cut is a set of edges $C \subset E$ such that the two terminals become separated on the induced graph $G'=(V,E \setminus C)$.

3.2 Max-Flow/Min-Cut Theorem

Graph cut segmentation is solved by the max-flow/min-cut theorem. However, we do not have to look too deep into the theorem to understand the concept of graph cut segmentation. Thus, here we first cover a few definitions related to the flow network, and then we state the max-flow/min-cut theorem.

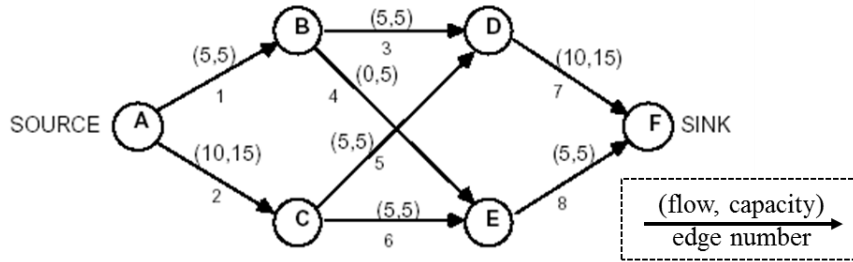


Fig. 3.1 An example of a flow network

A **flow network** is defined as a directed graph where an edge has a nonnegative capacity. The **capacity** of an edge is denoted by c_{uv} . It represents the maximum amount of flow that can pass through an edge. We denote a source terminal as s and a sink terminal as t . A **cut** (S, T) of graph G is a partition of V into S and $T = V \setminus S$ such that $t \in T$ and $s \in S$. A **flow** is denoted by f_{uv} , subject to the following two constraints:

1. $f_{uv} \leq c_{uv}$ for each $(u, v) \in E$ (capacity constraint)
2. $\sum_{u:(u,v) \in E} f_{uv} = \sum_{u:(v,u) \in E} f_{vu}$ for each $v \in V \setminus \{s, t\}$ (conservation of flows).

The **value of flow** is defined by $|f| = \sum_{v \in V} f_{sv}$, where s is the source of G . It represents the amount of flow passing from the source to the sink. The *maximum flow problem* is to maximize $|f|$, that is, to route as much flow as possible from s to t . An **s-t cut** $C = (S, T)$ is a partition of V such that $s \in S$ and $t \in T$. When we remove C , no flow is able to pass from the source to the sink. The **capacity of an s-t cut** is defined as

$c(S,T) = \sum_{(u,v) \in S \times T} c_{uv}$. The *minimum s-t cut problem* is minimizing $c(S,T)$, that is, to

determine S and T such that the summed capacity of the S - T cut is minimal.

The **max-flow min-cut theorem** proves that the maximum network flow and the sum of the cut-edge weights of any minimum cut that separates the source and the sink are equal.

With the basic ideas of a flow network, we can now move on to build the graph cut algorithm on top of it.

3.3 Graph Cut

Let an undirected graph be denoted as $G = (V, E)$. The vertex V is composed of two different kinds of nodes (vertices). The first kind of vertices is neighborhood nodes, which are composed of pixels in the image. The second kind of vertices is called terminal node. There are two terminal nodes added for applying max-flow/min-cut algorithm: an “object” terminal (a source S) and a “background” terminal (a sink). This kind of graph is also called *s-t graph*. In *s-t graph*, there are also two types of edges. The first type of edges is called n-links which connect the neighboring pixels within the image (Here we adopt 4-connected system in the 2D image). And the second type of edge is called t-links which connect the terminal nodes with the neighborhood nodes. In this kind of graph, each edge is assigned with a non-negative weight denoted as $e w$ which is also named as cost. The cost of the cut $|C|$ is the sum of the weights on edges C , which is expressed as $|C| = \sum_{e \in C} w_e$.

A minimum cut is the cut that have the minimum cost called min-cut and it can be achieved by finding the maximum flow which is verified in [12, 13, 23] that the min-cut

is equivalent to max-flow. The max-flow/min-cut algorithm developed by Boykov and Kolmogorov [23] can be used to get the minimum cut for the s-t graph. Thus, the graph is divided by this cut and the nodes are separated into two disjoint subsets S and T where $s \in S$, $t \in T$ and $S \cup T = V$. The two subsets correspond to the foreground and background in the image segmentation. This kind of graph can be depicted in Figure 3-1.

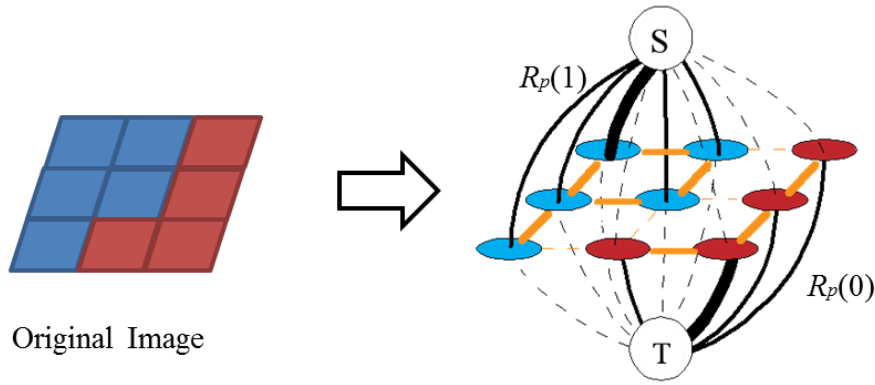


Fig. 3.2 Illustration of s-t graph. The image pixels correspond to the neighbor nodes in the graph(except s and t nodes). The orange lines in the graph are n-links and the black lines are t-links.

3.4 Graph Cut Segmentation

3.4.1 Basic Ideas and Background Information

Image segmentation can be regarded as pixel labeling problems. The label of the object (s-node) is set to be 1 while that of the background (t-node) is given to be 0 and this process can be achieved by minimizing the energy-function through minimum graph cut. In order to make the segmentation reasonable, the cut should be occurred at the boundary between object and the background. Namely, at the object boundary, the energy (cut) should be minimized. Let the labeled pixel set be $L = \{l_1, l_2, l_3, \dots, l_i, \dots,$

$l_n\}$ where n is the number of the pixels in the image and $l_i \in \{0,1\}$. Thus, the set L is divided into 2 parts and the pixels labeled with 1 belong to object while others are grouped into background.

Greig et al. [8] first proposed that the min-cut/max-flow algorithms from combinatorial optimization can be used to minimize certain important energy functions in vision. The energies addressed by Greig et al. and by most later graph-based methods can be represented as

$$E(L) = R(L) + \lambda \cdot B(L). \quad (3.1)$$

$R(L)$ is called the regional term, while $B(L)$ is the boundary term. The coefficient $\lambda \geq 0$ specifies a relative importance of $R(L)$ versus $B(L)$. The regional term $R(L)$ represents the individual penalties for assigning pixel p to “object” or “background”. It can be defined as

$$R(L) = \sum_p R_p(l_p) \quad (3.2)$$

where l_p is the label for foreground or background. $R_p(l_p)$ is the weight on a t-link edge. It can be obtained by comparing the intensity of pixel p with the given histogram of the object and background. Other features might as well be used to replace the intensity. We are designing a term that will be minimized in later computation process. Therefore, we should let the penalty of a pixel grouping into an area, e.g. $R_p(l_p)$, be smaller when p has some quality similar to the area. For instance, the weight of the t-links can be defined as following equations.

$$R_p(1) = -\ln \Pr(I_p|1) \quad (3.3)$$

$$R_p(0) = -\ln \Pr(I_p|0) \quad (3.4)$$

where $\Pr(I_p|1)$ is the conditional probability given the intensity information of pixel p

and the object. When $\Pr(I_p|1)$ is larger than $\Pr(I_p|0)$, $R_p(1)$ will be smaller than $R_p(0)$. This means when the pixel is more likely to be the object, the penalty for grouping that pixel into object should be smaller which can reduce the energy in (3.1). Thus, when all of the pixels have been correctly separated into two subsets, the regional term would be minimized.

On the other hand, the boundary term $B(L)$ comprises the “boundary” properties of segmentation. $B(L)$ can be defined as

$$B(A) = \sum_{\{p,q\} \in N} B_{p,q} \cdot \delta(l_p, l_q), \quad (3.5)$$

where p, q are adjacent pixels and $\delta(l_p, l_q)$ is defined as:

$$\delta(l_p, l_q) = \begin{cases} 1 & \text{if } l_p \neq l_q \\ 0 & \text{if } l_p = l_q \end{cases} \quad (3.6)$$

Each pair of neighboring pixels $\{p, q\}$ in N is connected by an n -link. Coefficient $B_{p,q} \geq 0$ is the weight on an n -link edge. It could be interpreted as a penalty for a discontinuity between p and q . Normally, $B_{p,q}$ is large when pixels p and q are similar (e.g. in their intensity) and $B_{p,q}$ is close to zero when the two are very different. The penalty $B_{p,q}$ can also decrease as a function of distance between p and q . Costs $B_{p,q}$ may be based on local intensity gradient, Laplacian zero-crossing, gradient direction or other criteria. Often, it is sufficient to set the boundary penalties from a simple function like

$$B_{p,q} = c \cdot \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{\text{dist}(p, q)} \quad (3.7)$$

where c is an arbitrary constant and σ can be estimated as “camera noise”.

With (3.2) and (3.5), we can rewrite (3.1) into

$$E(L) = \sum_p R_p(l_p) + \lambda \cdot \sum_{\{p,q\} \in N} B_{p,q} \cdot \delta(l_p, l_q) \quad (3.8)$$

In order to get a reasonable segmentation result, the assignment of the weight in the s-t graph is very important. Here we sum up the “design guideline” of the regional and boundary term:

1. For every pixel: when the feature of the pixel is inclined to be the object, the regional term between this pixel and s-node will be larger than that between pixel and t-node which means the cut is more likely occurred at the edge with smaller weight.
2. For the neighboring pixels, when their intensity is very similar, the boundary term should be large so that they are not likely to be separated by the cut.

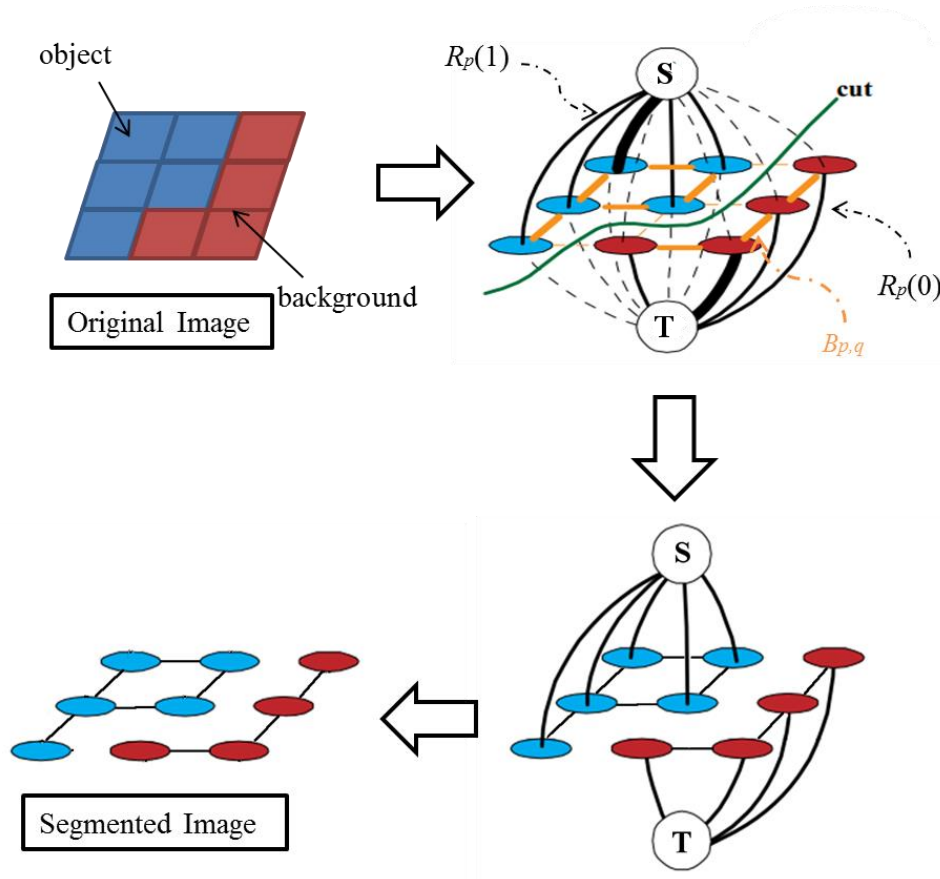


Fig. 3.3 Illustration of graph cut for image segmentation. The cut is corresponding to the minimal energy of (3.1)

3.4.2 Hard Constraints

So far we have covered the basic ideas of graph cuts as well as the essence of the energy function. Now we step back a little to combine the concept above with our user interaction.

In real examples, objects may not have sufficiently distinct regional properties. In such cases it becomes necessary to further constraint the search space of possible solutions before computing an optimal one. Boykov et al. proposed using topological (hard) constraints as an important source of “high level” contextual information about the object of interest in real images.

Assume that O and B denote the subsets of pixels *p priori* known to be a part of “object” and “background”, correspondingly. Naturally, the subsets $O \subset P$ and $B \subset P$ are such that $O \cap B = \emptyset$. Our goal is to compute the global minimum of (3.1) among all segmentations L satisfying “hard constraints”

$$\forall p \in O: l_p = 1 \quad (3.9)$$

$$\forall p \in B: l_p = 0 \quad (3.10)$$

The hard constraints can be used to initialize the algorithm and to edit the results. It can be set either automatically or manually depending on an application. Manually controlled seed is the most likely way to enter hard constraints in many generic applications. On the other hand, automatically set hard constraints can be used to initialize the algorithm in highly specialized applications such as organ segmentation from medical images or volumes.

In interactive segmentation framework, the user provides seeds to indicate foreground and background regions. The seeded regions are viewed as hard constraints, that is, they will be labeled accordingly. In addition, the information provided by seeded

region will be viewed as prior knowledge to help generate edge weights in graph cut segmentation.

In Boykov and Jolly's method, the weight of the s-t graph is given in Table 3-1.

Table 3.1 Weight of the s-t graph. P represent all the pixels in the original image and

$$K = 1 + \max_{p \in P} \sum_{q: \{p,q\} \in N} B_{p,q}.$$

edge	weight	For
$\{p, q\}$	$B_{p,q}$	$\{p,q\} \in N$
$\{p, S\}$	$R_p(0)$	$p \in P, p \notin O \cup B$ (unknown)
	K	$p \in O$
	0	$p \in B$
$\{p, T\}$	$R_p(1)$	$p \in P, p \notin O \cup B$ (unknown)
	0	$p \in O$
	K	$p \in B$

At the end of this subsection, we use a simple example to demonstrate a more detailed process of graph cut segmentation.

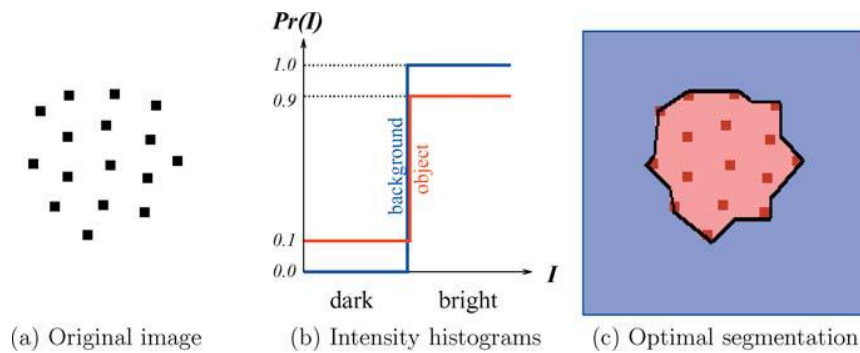
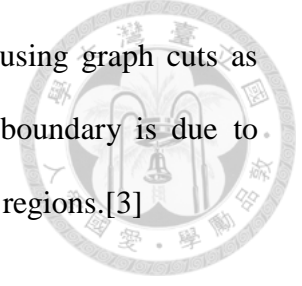


Fig. 3.4 The optimal object segment (red tinted area in (c)) finds a balance between

“region” and “boundary” terms in (2). The solution is computed using graph cuts as explained in Section 2.5. Some ruggedness of the segmentation boundary is due to metrication artifacts that can be realized by graph cuts in textureless regions.[3]



In Fig. 3.4 illustrates some interesting properties of the cost function(3.1). The object of interest is a cluster of black dots in Fig. 3-3(a) that we would like to segment as one blob. We combine boundary term (3.5) and region term (3.2) and take $\lambda > 0$ in (3.1). The penalty for discontinuity in the boundary cost is defined manually as:

$$B_{p,q} = \begin{cases} 1 & \text{if } I_p = I_q \\ 0.2 & \text{if } I_p \neq I_q \end{cases} \quad (3.11)$$

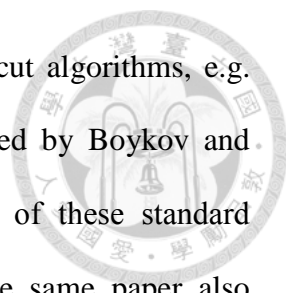
To describe regional properties of segments we use a priori known intensity histograms (Fig. 3-3(b)). Note that the background histogram concentrates exclusively on bright values while the object allows dark intensities observed in the dots. If these histograms are used in (3.2, 3.5) then we get the following regional penalties $R_p(I_p)$ for pixels of different intensities.

Table 3.2 Manually assigned regional values.

I_p	$R_p(1)$	$R_p(0)$
dark	2.3	$+\infty$
bright	0.1	0

The optimal segmentation in Fig. 2.2(c) finds a balance between the regional and the boundary term of energy (2.1). Individually, bright pixels slightly prefer to stay with the background (see Table 3.2). However, boundary term forces some of them to agree with nearby dark dots.

Note that a fast implementation of graph cut algorithms can be an issue in practice.



The most straight-forward implementations of the standard graph cut algorithms, e.g. max-flow or push-relabel, can be slow. The experiments conducted by Boykov and Kolmogorov [4] compared several well-known “tuned” versions of these standard algorithms in the context of graph based methods in vision. The same paper also describes a new version of the max-flow algorithm that significantly outperformed the standard techniques. We adopted this algorithm as the base of our segmentation scheme.

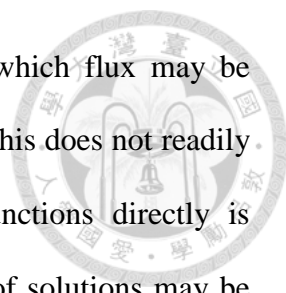
3.5 Issues in Graph Cut Algorithm

3.5.1 The Shrinking Bias

The *shrinking bias* refers to the tendency of graph cut algorithm towards shorter paths. (This length bias predates graph-cut methods and is present in earlier least-cost path approaches [13].) The tendency is caused by the boundary term, which consists of a summation over the boundary of the segmented regions.

This can be especially noticeable when these methods shortcut across the interior of an object to avoid segmenting an appendage as illustrated in Fig. 3.5. This is offset somewhat by the region term, which tries to penalize shortcuts through areas where the labeling is clear from the coloring. However, this is not without tradeoffs—overweighting the region term to compensate for the boundary term’s shrinking bias can result in discontinuous objects with coloring similar to the user’s respective scribbles being incorrectly selected (as also illustrated in Fig. 3.5). This leads to a delicate balance between the weighting of the two terms and the resulting strengths and limitations of each.

Most approaches for otherwise avoiding the shrinking bias in graph-cut and similar approaches involve variations on normalizing the cost of the cut by the size of the



resulting object(s). This may be done for grey-level images for which flux may be defined along the boundary of the region [15], but as noted in [16], this does not readily extend to color images. Optimizing general normalized cost functions directly is NP-hard but may be approximated [17]. Alternatively, a subspace of solutions may be explored by varying the relative weighting of the boundary and region terms [18]. (The authors of [18] note, however, that this method is not fast enough for interactive color image segmentation.) More recent work in [19] uses a curvature-minimizing rather than length-minimizing regularization term to smooth the resulting boundaries while avoiding shortcutting, but this does not use an edge component to localize edges, nor does it run in interactive time.

Because of the complementary strengths and weaknesses of seed-expansion and graph-cut approaches, some have suggested combining them. Work in [20] showed that graph-cuts and random-walkers [21] (a form of seed expansion), along with a new method similar in principle to geodesic segmentation [22], could be placed in a common framework in which the three methods differ by only the (integer) selection of a single parameter. The idea is further expanded in [25] by varying a second parameter. They also showed empirically that of these three approaches the new method (analogous to geodesic segmentation) is most sensitive to seed placement while “because of the ‘small cut’ phenomenon, the Graph Cuts segmentation is the least robust to the quantity of seeds.” More recent work [26] has explored a way to blend the respective strengths of these methods using non-integer selections for the free parameter in [20], determining a suitable parameter selection empirically over a set of sample images with known ground truth.

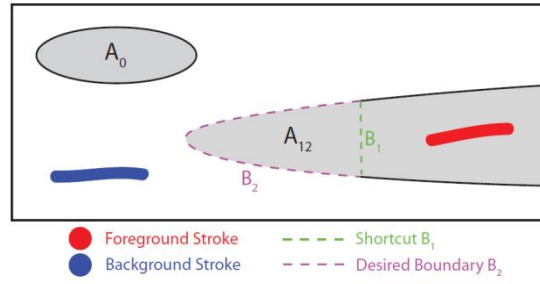


Fig. 3.5 Illustration of shrinking bias. A_{12} and A_0 are some homogeneous areas. Graph-cut methods may short-cut across the desired object along B_1 instead of following the true edge B_2 because less cost is accrued traversing the shorter path.

3.5.2 Efficiency

Mean reaction time for college-age individuals is approximately 190 milliseconds to detect visual stimulus. To our knowledge, the technique proposed by Boykov et al.[3] is the most efficient implementation of max-flow/min-cut algorithm to date. Unfortunately, as shown in the last column of Table 1[5], the max-flow algorithm fails to provide interactive visual feedback for real life image cutouts. To improve efficiency, [5] proposed building the graph cut formulation on a pre-segmented image instead of image pixels. Because our framework is inspired by [5], we will introduce the algorithm in details in 3.7.

3.5.3 Vulnerability to Noise

A small amount of noise (adjusting even a single pixel) could cause the contour returned by the algorithm to change drastically. This is caused by the same nature that causes the shrinking bias.

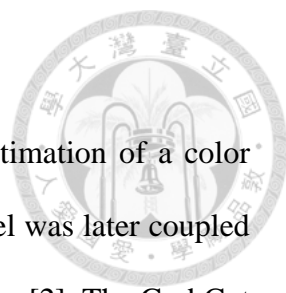
3.6 Prior Works

The graph cuts segmentation algorithm has been extended in three different directions in order to address issues of different factors. We briefly introduce the three directions in this section.

3.6.1 Speed-up based graph cut

The first type of extension to the graph cuts algorithm has focused on increasing speed. In order to speed up the computation of graph cut algorithm, implementation based on GPU with CUDA code is proposed in [53]. This kind of methods increase speed with the parallel computing, which perform well compared to sequentially computing. However, the most used method for reducing the computational time for the graph cut related algorithms is coarsening the graph before applying the graph cuts algorithm. This coarsening has been accomplished in two manners: **1)** by applying a standard multilevel approach and solving subsequent, smaller graph cuts problems in a fixed band to produce the final, full-resolution segmentation [12]. **2)** by applying a watershed algorithm to the image and treating each watershed basin as a “supernode” in a coarse graph to which graph cuts is applied [5]. The primary goal of these two approaches is to increase the computational speed of graph cuts by intelligently reducing the number of nodes in the graph. As stated in [12], the objective is to produce the same segmentation result as regular graph cuts by introducing a heuristic that greatly speeds the computation. Therefore, the benefits and difficulties of the graph cuts algorithm listed above also apply to these approaches, with an added uncertainty about the role of the coarsening operator in the final result (i.e., the final segmentation is no longer guaranteed to be the minimum cut).

3.6.2 Interactive-based graph cut



The second direction of extension started from an iterative estimation of a color model with the graph cuts algorithm in [9]. This iterative color model was later coupled with an alteration of the user interface to create the GrabCut algorithm [2]. The GrabCut approach asks the user to draw a box around the object to be segmented and employs the color model as priors (“t-links”) to obviate the need for explicit specification of foreground seeds. The added color model is of clear value in the application of color image segmentation and the “boxinterface” requires less user interaction. Although the approach does perform well in the domain of color image segmentation, the iterative nature of the algorithm does increase the computational burden of the algorithm (requiring a solution to the max-flow/min-cut problem in every iteration). In addition, there is no longer a guarantee of optimality (the algorithm is terminated when the iterations stagnate). The “box-interface” is not always sufficient to capture the desired object, since further editing of the results with standard graph cuts is often required.

3.6.3 Shape prior-based graph cut

Due to noise, diffuse edge or occluded objects, conventional graph cut which only incorporate regional and edge information cannot get ideal segmented object. Even though the interactive-based graph cut can reduce these problems to some extent, many rounds of interaction will be needed and this will affect the segmentation efficiency. Thus, the shape prior-based graph cut algorithms are widely researched. They incorporate the shape information of the object into the energy function to improve the segmentation result [32]-[36]. Especially for the images with known prior information, shape prior-based graph cut can work well when the shape is described appropriately. In

most of the cases [32]-[36], the energy of the shape prior is combined into the energy function. The energy function for the shape prior-based graph cut is usually defined as following equation:

$$E(L) = R(L) + B(L) + E_{shape} \quad (3.12)$$

Where, $R(L)$ is the regional term, $B(L)$ is the boundary term and E_{shape} is the prior shape term. The energy for the regional and boundary term can be described as that in section 3.4. For the energy in the shape term, the aim is to make it smaller for the pixels around the object boundary and bigger for that far away from the boundary of object. There are two kinds of energy representation for the shape prior. The first one computes the weights of n-links with the gradient and shape prior information, while the weights of the t-links are obtained as the traditional way in section 3.4. The second one, on the opposite, reflects the shape prior information in the regional term (t-links), while the boundary term (n-links) are obtained by the way in section 3.4.

For the shape prior-based graph cut, the establishment of the shape template is very important. Many pre-segmented object will be adopted to be the training set to build the shape template. Furthermore, the alignment of the training set is also inevitable for making the shape template fit the segmented image better.

3.7 Lazy Snapping[5]

We would spend some time introducing Lazy Snapping[5] because our work is primarily based on it. Fig. 3.6 shows how the interactive image cutout system works.

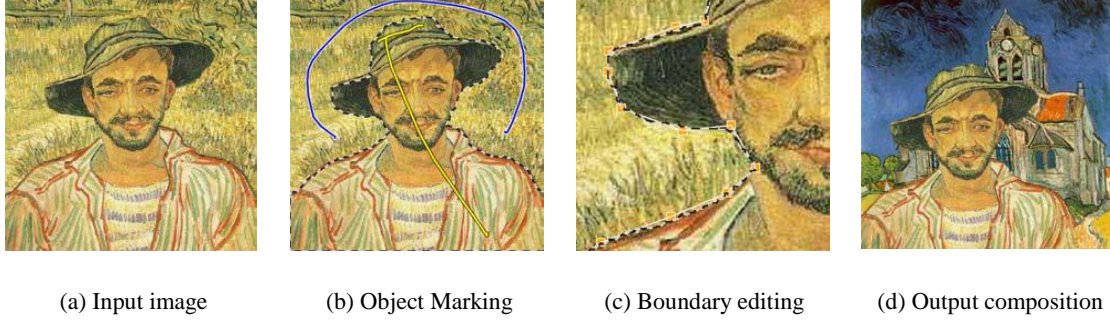


Fig. 3.6 Lazy Snapping is an interactive image cutout system, consisting of two steps: a quick object marking step and a simple boundary editing step. In (b), only 2 (yellow) lines are drawn to indicate the foreground, and another (blue) line to indicate the background. All these lines are far away from the true object boundary. In (c), an accurate boundary can be obtained by simply clicking and dragging a few polygon vertices in the zoomed-in view. In (d), the cutout is composed on another Van Gogh painting.

Lazy Snapping consists of two steps: an *object marking* step and a *boundary editing* step. The first step, object marking, works at a coarse scale, which specifies the object of interest by a few marking lines. The second step, boundary editing, works at a finer scale or on the zoomed-in image, which allows the user to edit the object boundary by clicking and dragging polygon vertices.

Here we only introduce the first step in details. As introduced earlier in section 3.4, the general energy form we try to minimize is
$$E(L) = \sum_i R_i(l_i) + \lambda \cdot \sum_{\{i,j\} \in N} B_{i,j} \cdot \delta(l_i, l_j) \quad ,$$

where $R_i(l_i)$ is the regional term, encoding the cost when the label of node i is l_i , and

$B_{i,j} \cdot \delta(l_i, l_j)$ is the boundary term, encoding the cost when the labels of adjacent nodes i and j are l_i and l_j respectively.

To compute R , first the colors in foreground seeds and background seeds (marked by the user) are clustered by the K-means method. The mean colors of the foreground and background clusters are denoted as $\{K_n^F\}$ and $\{K_m^B\}$ respectively. The K-means method is initialized to have 64 clusters. Then, for each node i , we compute the minimum distance from its color $C(i)$ to foreground clusters as $d_i^F = \min_n \|C(i) - K_n^F\|$, and to background clusters as $d_i^B = \min_m \|C(i) - K_m^B\|$. Therefore, $R_i(l_i)$ is defined as follows:

Table 3.3 Definitions of $R_i(l_i)$

	$l_i = 1$	$l_i = 0$
$i \in F$	0	∞
$i \in B$	∞	0
$i \in U$	$\frac{d_i^F}{d_i^F + d_i^B}$	$\frac{d_i^B}{d_i^F + d_i^B}$

where F and B denote foreground and background respectively, and $U = V \setminus \{F \cup B\}$ (the uncertain region).

We define B as a function of the color gradient between two nodes i and j :

$$B(i, j) = |l_i - l_j| \cdot g(C_{ij}) \quad (3.13)$$

where $g(\xi) = \frac{1}{\xi + 1}$, and $C_{ij} = \|C(i) - C(j)\|^2$ is the L2-Norm of the RGB color

difference of two pixels i and j . Again, the boundary term B represents the penalty for

boundaries. It is larger when adjacent pixels have similar color. With all definitions in place, we then minimize the energy $E(X)$ using the max-flow/min-cut algorithm[3].

To improve efficiency, Li et al. introduced a novel graph cut formulation which is built on a pre-computed image over-segmentation instead of image pixels. They chose the watershed algorithm [37] as their pre-segmentation algorithm. As shown in Figure 3, we can use the same notation $G = (V, E)$ for the new graph, while the nodes V are the set of all small regions from pre-segmentation, and the edges E are the set of all arcs connecting adjacent regions.

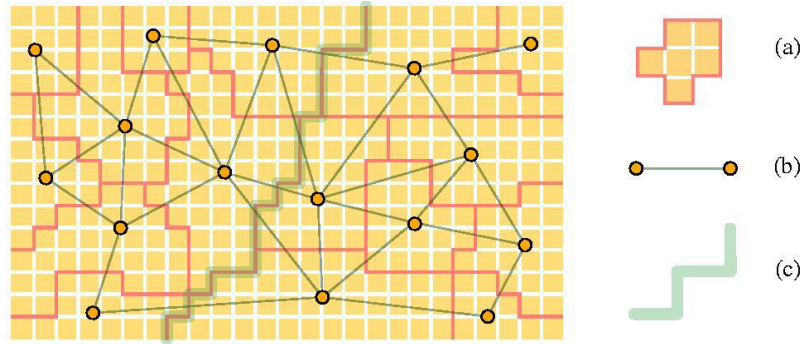


Fig. 3.7 (a) A small region by the pre-segmentation. (b) The nodes and edges for the graph cut algorithm with pre-segmentation. (c) The boundary output by the graph cut segmentation.

Li et al. did not release source code or software of their work, so we are unclear of their exact framework. Nevertheless, we present a complete algorithm in Table 3.4 for reference.

Table 3.4 Core algorithm of first step in Lazy Snapping

Algorithm Lazy Snapping

Input: Original image **I**, user labeled seeds **B** and **F**

Output: Foreground/background labels **L**

- 1: Perform watershed segmentation on **I**
 - 2: Extract foreground segments **F** and background segments **B**
 - 3: Compute $\{K_n^F\}$ and $\{K_n^B\}$ with **B** and **F**
 - 4: **for** each superpixel **i do**
 - 5: Assign boundary and regional term with Table 5.1 and (5.5)
 - 6: **end for**
 - 7: Solve **L** by Max-Flow/Min-Cut Optimization
-

Chapter 4 Overview of Superpixel Methods

In this chapter, we first introduce the concept and possible applications of superpixels in section 4.1. Two main categories of prior works are reviewed in section 4.2 and section 4.3. Then, we give a summary of superpixel methods in section 4.4. Finally, we introduce the benchmark proposed by Neubert and Protzel[43] and present the evaluation results of methods we introduced in section 4.2 and section 4.3.

4.1 Introduction

A superpixel is commonly defined as a perpetually uniform region in the image. The major advantage of using superpixels is computational efficiency—a superpixel representation greatly reduces the number of image primitives compared to the pixel representation. It has been proved useful for applications such as depth estimation, image segmentation, body model estimation and object localization.

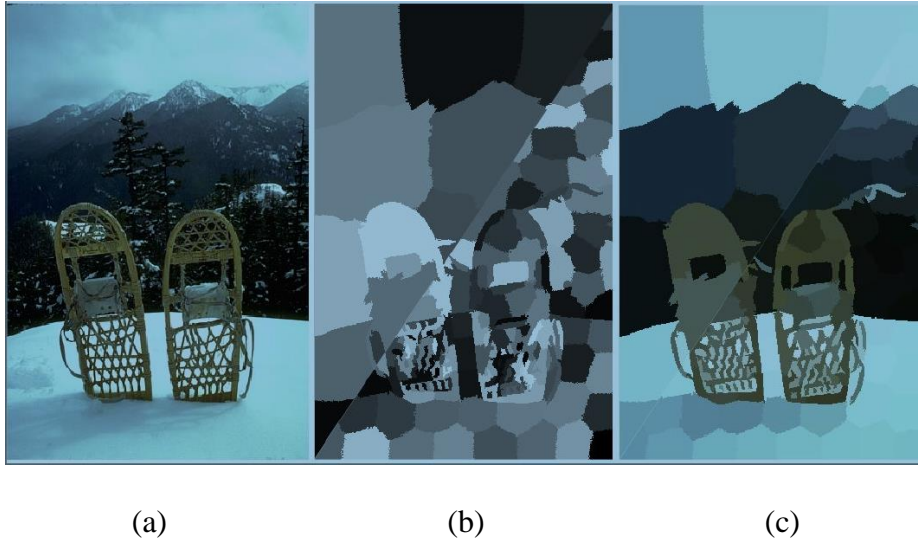
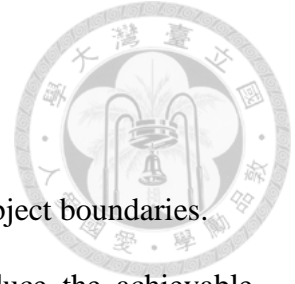


Fig. 4.1 An superpixel segmentation example. The input image (a) is segmented into 25 (middle part, top-left corner) and 250 (middle part, bottom-right) superpixels in (b). The segmentation result is shown with mean colors of each region in (c).

Generally, superpixel contains the following properties [38]:

- Every superpixel should overlap with only one object.
- The set of superpixel boundaries should be a superset of object boundaries.
- The mapping from pixels to superpixels should not reduce the achievable performance of the intended application.
- The above properties should be obtained with as few superpixels as possible.

We can broadly classify superpixel algorithms into graph-based and gradient-ascent-based methods. In graph-based algorithms, each pixel is treated as a node in a graph, and edge weight between two nodes are set proportional to the similarity between the pixels. Superpixel segments are extracted by minimizing a cost function defined on the graph. Gradient-ascent-based methods start from an initial rough clustering, and then refine the clusters from the previous iteration with gradient ascent methods to obtain better segmentation until convergence.



4.2 Gradient-Ascent-Based Algorithms



1. Watershed (WS)

The watershed approach [44] performs a gradient ascent starting from local minima to produce watersheds, lines that separate catchment basins. The resulting superpixels are often highly irregular in size and shape, and do not exhibit good boundary adherence. The approach of [44] is relatively fast ($O(N \log N)$ complexity), but does not offer control over the amount of superpixels or their compactness.

2. Meanshift (MS)

Mean-shift [47] (MS) is a mode-seeking algorithm that generates image segments by recursively moving to the kernel smoothed centroid for every data point in the pixel feature space, effectively performing a gradient ascent [46]. The generated segments/superpixels can be large or small based on the input kernel parameters, but there is no direct control over the number, size, or compactness of the resulting superpixels.

3. Quickshift (QS)

Quick shift [46] is also a mode-seeking segmentation scheme like Mean-shift, but is faster in practice. It moves each point in the feature space to the nearest neighbor that increases the Parzen density estimate. While it has relatively good boundary adherence, QS is quite slow, with an $O(dN^2)$ complexity (d is a small constant [47]). QS does not allow for explicit control over the size or number of superpixels.

4. Turbopixel (TP)

The Turbopixel method progressively dilates a set of seed locations using level-set based geometric flow [47]. The geometric flow relies on local image gradients, aiming to regularly distribute superpixels on the image plane. Unlike WS, TP superpixels are constrained to have uniform size, compactness, and boundary adherence. TP relies on algorithms of varying complexity, but in practice, as the authors claim, has approximately $O(N)$ behavior [47]. However, it is among the slowest algorithms examined and exhibits relatively poor boundary adherence.

5. Simple Linear Iterative Clustering (SLIC)

The SLIC algorithm begins with sampling K regularly spaced cluster centers and moving them to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood [48]. The clustering is based on superpixels' color similarity and proximity in the image plane (5-dimensional space $[labxy]$). Since the spatial distance in the xy plane depends on the image size, Achanta et al. propose a new distance measure to normalize the distance term. This measure ensures superpixels have similar cluster sizes and also regular shapes. SLIC achieves $O(N)$ complexity and is among the fastest algorithms.

4.3 Graph-Based Algorithms

1. Normalized Cuts (NC)

The Normalized cuts algorithm [49] recursively partitions a graph of all pixels in the image using contour and texture cues, globally minimizing a cost function defined on the edges at the partition boundaries. It produces very regular, visually pleasing superpixels. However, the boundary adherence of NC is relatively poor and it is the slowest among the methods (particularly for large images), although attempts to speed up the algorithm exist [50]. NC has a complexity of $O(N^{\frac{3}{2}})$ [49], where N is the number of pixels.

2. Felzenszwalb and Huttenlocher (FH)

Felzenszwalb and Huttenlocher [51] propose an alternative graph-based approach that has been applied to generate superpixels. It performs an agglomerative clustering of pixels as nodes on a graph, such that each superpixel is the minimum spanning tree of the constituent pixels. FH adheres well to image boundaries in practice, but produces superpixels with very irregular sizes and shapes. It is $O(N \log N)$ complex and fast in practice. However, it does not offer an explicit control over the amount of superpixels or their compactness.

3. Entropy Rate Superpixel (ERS)

Liu et al. propose solving the superpixel segmentation problem by maximizing an objective function on the graph topology [52]. The objective function consists of an entropy rate and a balancing term for obtaining superpixels with similar sizes. They also present a greedy optimization scheme to reduce the complexity of the algorithm to



$O(N \log N)$ [52]. ERS offers control over the amount of superpixels and also their compactness.



4.4 Remarks

Table 4.1 Summary of existing superpixel algorithms.

	Graph-based			Gradient-ascent-based				
	NC [49]	FH [51]	SLIC [48]	WS [44]	MS [45]	QS [46]	TP [47]	ERS [52]
Complexity	$\frac{3}{N^2}$	$N \log \Lambda$	N	$N \log N$	N^2	dN^2	N	$N \log N$
Parameters	1	2	1	1	3	2	1	2
Superpixel number control ^a	Yes	No	Yes	No	No	No	Yes	Yes
Superpixel compactness control	No	No	Yes	No	No	No	No	Yes

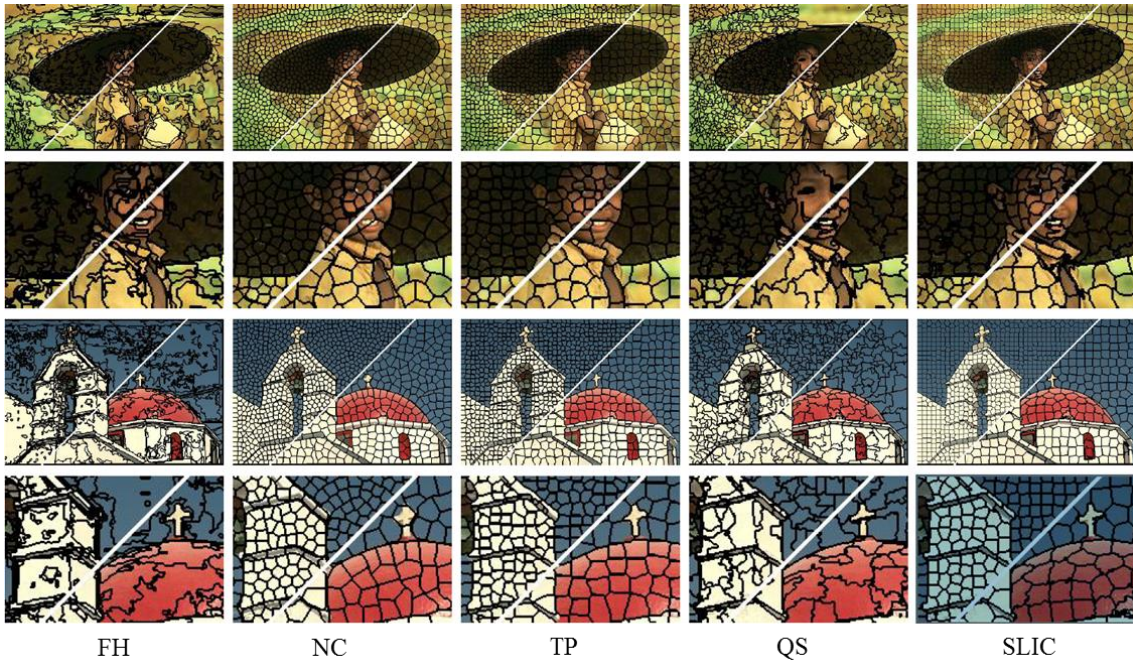


Fig. 4.2 Visual comparison of superpixels produced by various methods. The average superpixel size in the upper left of each image is 100 pixels, and 300 in the lower right. Alternating rows show each segmented image followed by a detail of the center of each image. [48]

4.5 Benchmark and Comparison

4.5.1 Benchmark

In [43], Neubert and Protzel proposed a standardized evaluation scheme for superpixel algorithms. We present their proposed dataset, user guidelines and error metrics below.

- **Dataset:** The image data base is BSDS500 [54], the extended version of the established Berkeley Segmentation Data Set. To ensure comparable results, the usage of the partitions of the BSDS 500 dataset is intended as follows:
 1. *Train* : There are 200 images for learning purposes.
 2. *Val*: The cross-validation dataset contains 100 images for adjusting the parameters of an algorithm.
 3. *Test* : 200 test set images are for the final benchmark.
- **Boundary Recall:** Boundary recall measures the percentage of the natural boundaries recovered by the superpixel boundaries. It is defined as the fraction of ground truth edges that fall within a certain distance d of a least one superpixel boundary. The benchmark presented in [43] was generated with $d = 2$. Given a ground truth boundary image G and the algorithms boundary image B , the computation of boundary recall is as followed:
 1. True Positives (TP): Number of boundary pixels in G for whose exist boundary pixel in B in range d .
 2. False Negatives (FN): Number of boundary pixels in G for whose does ot exist a boundary pixel in B in range d .
 3. Boundary Recall: $R = \frac{TP}{TP + FN}$



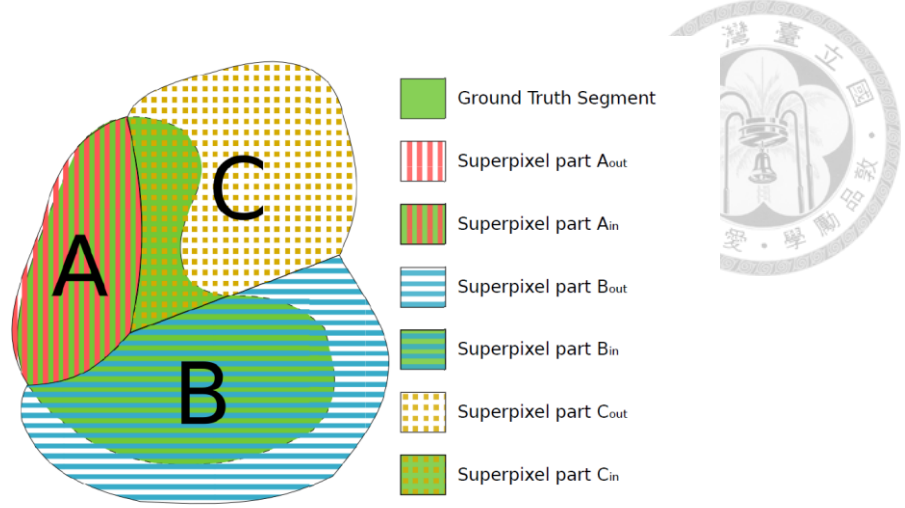


Fig. 4.3 Illustration of undersegmentation error. A ground truth segment (green) is covered by three superpixels (A,B,C), that can flood over the groundtruth segment border.[43]

- **Undersegmentation Error:** compares segment areas to measure to what extend superpixels flood over the ground truth segment borders. A ground truth segment divides a superpixel P in an *in* and an *out* part. This is illustrated in Fig. 4.3. There exist various implementations of undersegmentation error metrics. In [11], for each segment S it is summed over the out-parts of all superpixels P that overlap the segment:

$$e = \sum_{S \in GT} \frac{\sum_{P: P \cap S \neq \emptyset} |P_{out}|}{|S|} \quad (4.1)$$

For the example of Fig. 4.3, this is: $\frac{|A_{out}| + |B_{out}| + |C_{out}|}{|S|}$. However, there is a

serious penalty for large superpixels that have only a small overlap with the ground truth segment. Thus, Neubert and Protzel proposed a new formulation of the oversegmentation error and define the remaining error as the smaller error

introduced by either appending the *out* part to the segment or by omitting the *in* part of the superpixel. Being N the total number of pixels,

$$UndersegmentationError = \frac{1}{N} \left[\sum_{S \in GT} \left(\sum_{P: P \cap S \neq \emptyset} \min(P_{in}, P_{out}) \right) \right] \quad (4.2)$$

The inner sum is the error introduced by this specific combination of ground truth

segment and superpixel. In Fig. 4.3, this is $\frac{|A_{out}| + |B_{out}| + |C_{in}|}{|S|}$

4.5.2 Comparison Results

Algorithms tested are Normalized Cuts(NC) [49]¹, Felzenszwalb-Huttenlocher Segmentation(FH) [51]², Mean Shift (MS) [45]³, Quickshift(QS) [46]⁴, Marker-Controlled Watershed Segmentation (WS) [44]⁵, Entropy Rate Superpixel Segmentation (ERS) [52]⁶, Turbopixel Segmentation (TP) [47]⁷ and Simple Linear Iterative Clustering(SLIC) [48]⁸.

¹ <http://www.timotheecour.com/software/ncut/ncut.html>

² <http://www.cs.brown.edu/~pff/segment/>

³ <http://www.wisdom.weizmann.ac.il/~bagon/matlab.html>

⁴ <http://www.vlfeat.org/>

⁵ We use the OpenCV implementation with uniformly distributed markers
<http://opencv.willowgarage.com/wiki/>

⁶ <http://www.umiacs.umd.edu/~mingyliu/research.html#ers>

⁷ http://www.cs.toronto.edu/~babalex/turbopixels_supplementary.tar.gz

⁸ http://ivrg.ep.ch/supplementary_material/RK_SLICSuperpixels/index.html

1. Runtime

The runtime comparison given in Table 4.2 was carried out on an Intel Core 2. Quad Q9400 (2.66 GHz) processor with 4 GB RAM [43]. WS is by far the fastest algorithm, followed by FH and SLIC. The runtime of NC is too long, so the second half of the results are not displayed. To note that the range of runtimes of the different algorithms covers five orders of magnitudes.

2. Segmentation Quality

ERS produces similar-sized superpixels that fit the object boundary well. Around 500 segments ERS has the lowest undersegmentation error and the highest boundary recall among the methods. Further, ERS allows user to appoint an exact number of segments. Superpixel borders of FH strictly follow image gradients. It has high boundary recall, but also produces high undersegmentation error, especially when number of segments is small. The resulting segments vary strongly in shape and size. MS segments also vary in size and shape, but are more regular when compared with FH. Further, increasing the number of segments gives a good refinement of the prior segmentation. NC does not give such an refinement, but the superpixels are regularly sized, shaped and distributed. QS produces more irregular shapes and there occur small artifact segments. The regularity of segment borders of SLIC strongly depend on the compactness parameter. Tuning this parameter is crucial. However, the compactness parameter enables adjusting the regularity of the superpixels at the cost of losing some object borders. For WS, the resulting segment size and shape are not regulated. The quantitative benchmark results on segmentation quality are shown in Fig. 4.5 and Fig. 4.6. For boundary recall, SLIC performs very well on small numbers of segments, however, for larger numbers, FH and ERS achieve higher recall. SLIC performs well for

larger numbers compared to small numbers. This also follows the intuition, that segmentations with more irregular, longer border achieve higher boundary recall. For undersegmentation error, ERS, MS and SLIC perform best. TP performs badly on small numbers of superpixels and does not achieve the performance from the origin paper [11].

3. Visual Comparison

In Fig. 4.7, each row shows results of one algorithm with 2 parameter settings, visualized as overlayed boundary image and superpixels with average image colors. The algorithms parameters where chosen to produce about 100 and 500 superpixels (except for TP⁹ and NC¹⁰). The name of the algorithm name and the number of superpixels are given on the very left.

⁹ TP segmentation is based on growing of initial small segments. As noted in [43], for small numbers (e.g. 100 or 1000) of initial segments, these initial segments do not grow enough to cover the complete image. We consider this implementation as broken for small numbers of segments.

¹⁰ NC failed to segment and produced “out of memory” errors.

Table 4.2 Runtime (in seconds) of segmentation algorithms[43].

#superpixel algorithm	50	100	500	1000
NC	150	295	-	-
FH	0.13	0.13	0.13	0.13
SLIC	0.19	0.2	0.19	0.2
WS	0.01	0.01	0.01	0.01
MS	4.4	4.4	4.1	4
QS	9.25	5.86	3.05	2.71
TP	-	38.7	41.1	43.71
ERS	1.85	1.93	2.37	2.52

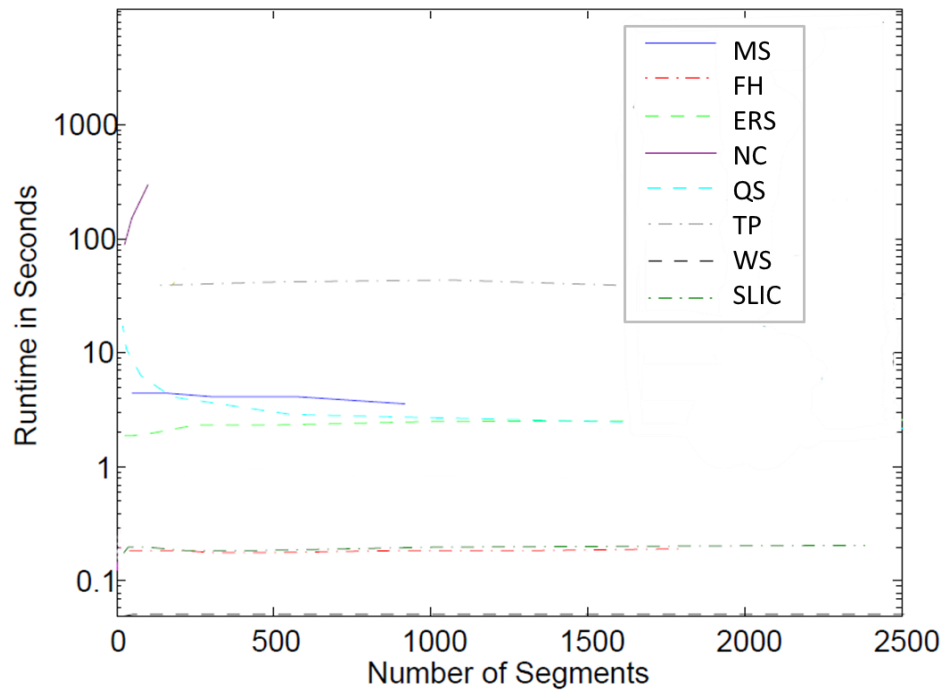


Fig. 4.4 Runtime of segmentation algorithms (log-scale) [43].

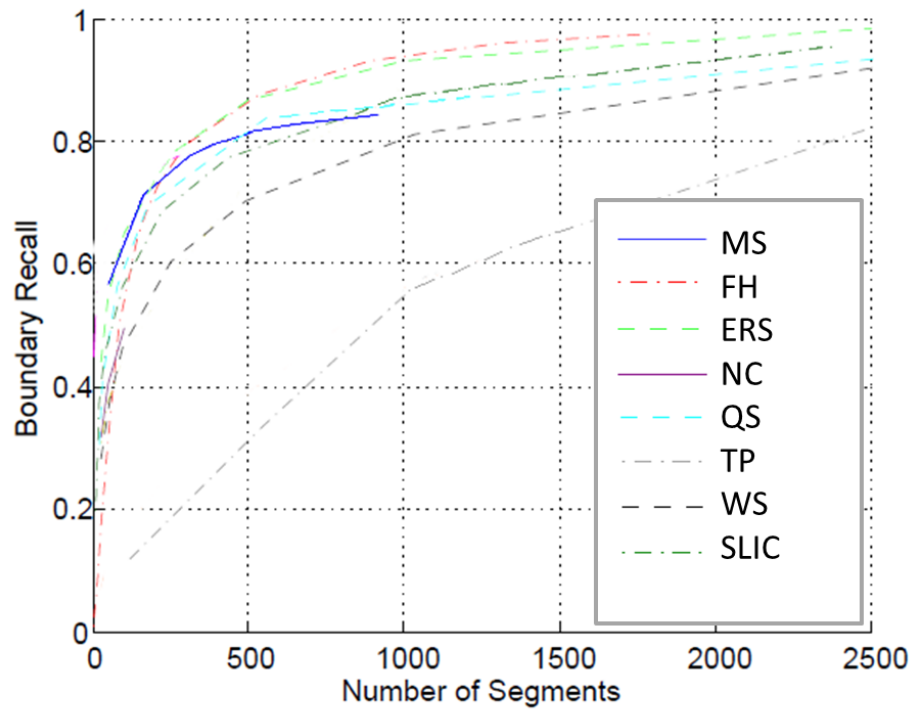


Fig. 4.5 Boundary recall of segmentation algorithms on Neubert and Protzel's benchmark(top-left is better).[43]

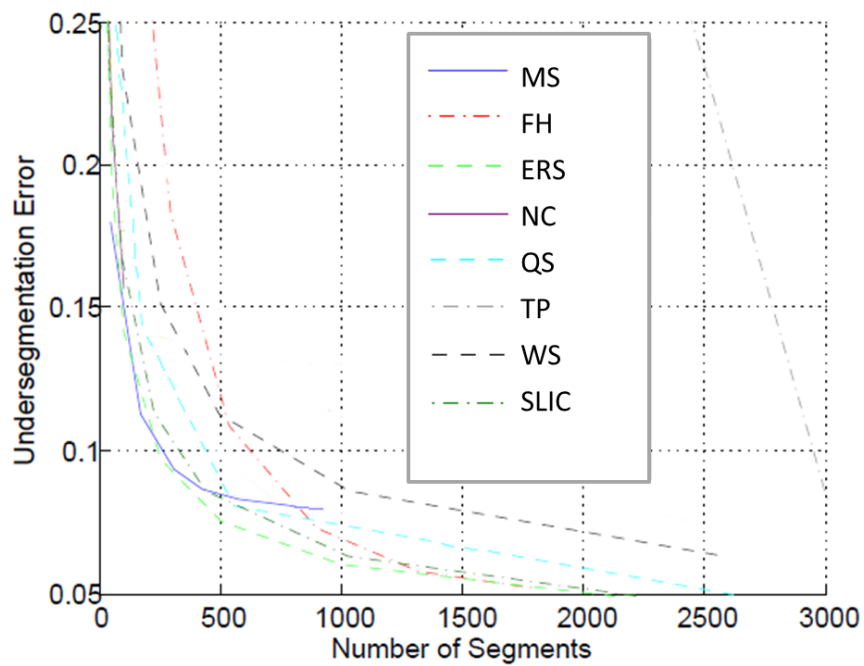


Fig. 4.6 Undersegmentation error of segmentation algorithms on Neubert & Protzel's benchmark(bottom-left is better). [43]

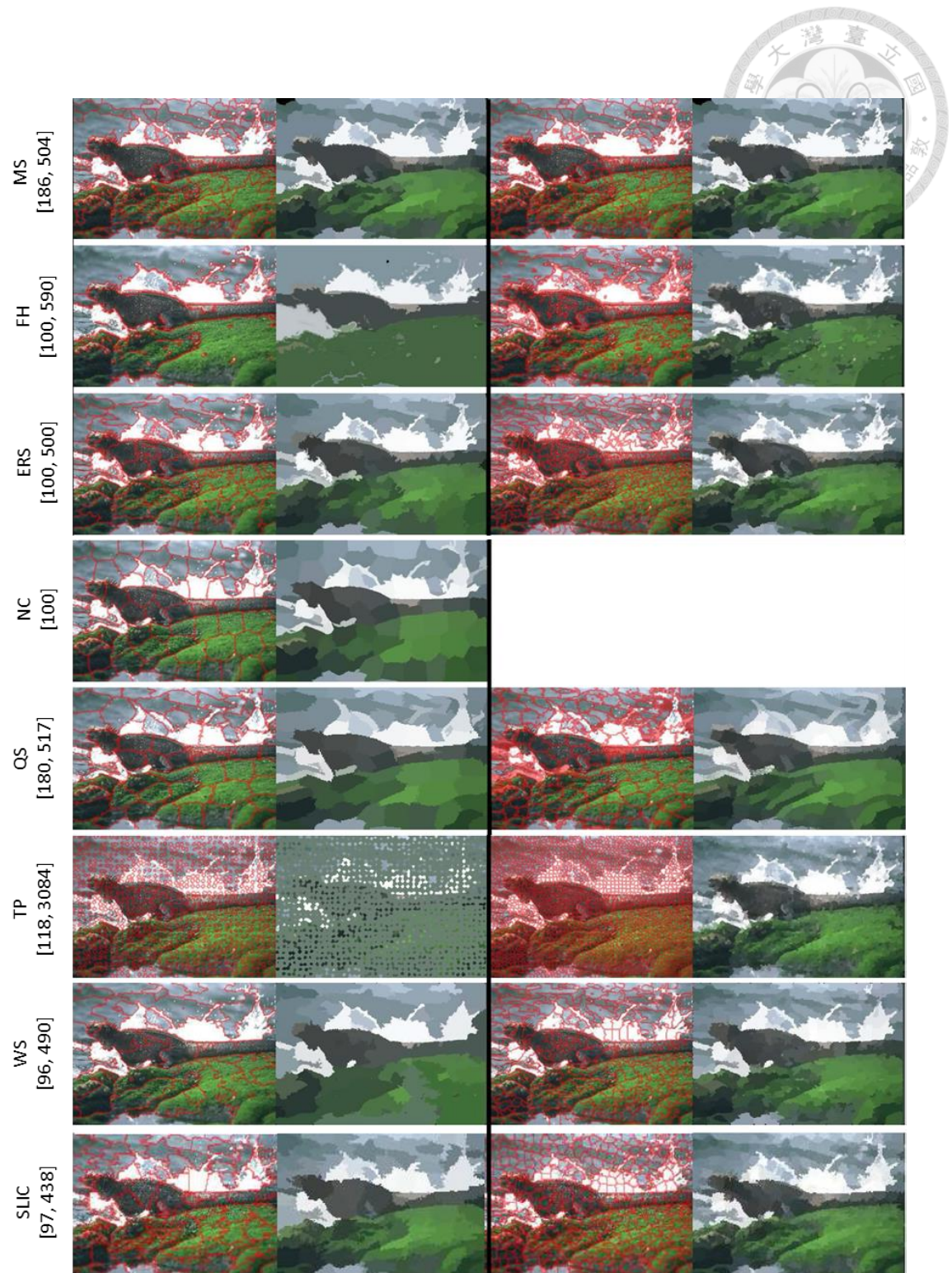


Fig. 4.7 Visual Comparison of superpixel segmentation results. The numbers on the left side indicate the number of superpixels in one image.

Chapter 5 Proposed Interactive Image Segmentation

Method



We have discussed the formulation of graph cut segmentation in details and also give a broad comparison on superpixel methods. The performance of graph cut based methods are held back by the Although many extensions on graph cut segmentation have been proposed, they are still not efficient enough to generate real-time result.

5.1 Observation

Observe the user experience of an interactive segmentation tool, we notice that the actual time delay the user experiences is the period from the user finishes marking objects to the segmentation result is displayed. Follow the typical framework of graph cut segmentation in section 3.4. The time delay is composed of computing energy terms and graph cut segmentation two parts. That means if we want to build a real-time segmentation tool, we should focus on cutting down the processing time of these two parts. One straightforward way is to incorporate superpixel pre-segmentation proposed in [5]. If we segment the image into superpixels and extract needed features.

Several applications have adopted watershed as their pre-segmentation method because of its speed. However, based on the theory above, if pre-segmentation can be done before the user finishes interaction, we don't have to count in the time spent on pre-segmentation as time delay. Assume it takes one second for the user to draw one scribble for object and another for background, the user will not feel any difference as long as the pre-segmentation process finishes under a second.

Nowadays users are more and more critical about the application response time. In [55], any response quicker than 100ms tends to be perceived as “instant”. Thus,

response time of the interactive system should aim at shorten the response time to 100ms or under.



5.2 User Interface (UI) Design

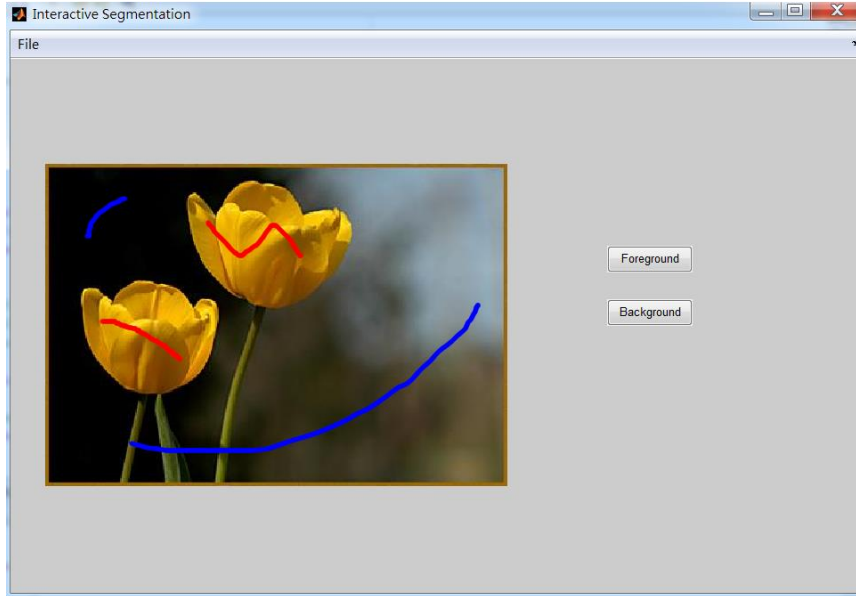


Fig. 5.1 Prototype of user interface.

We follow the “scribble-based” interactive tool UI design which is popular in recent years. To specify an object, a user marks a few lines on the image by dragging the mouse cursor while holding a button. A red line or a blue line is displayed for the foreground marker or background marker respectively. This high level, painting-type UI does not require very precise user inputs. As shown in Fig. 5.1, most marking lines are in fact far from the object boundary. After each marking, the user then inspects the segmentation result on screen and decides if more lines need to be marked. The response time is counted from the moment the user release the mouse button after each marking line is drawn to the segmentation result is displayed. It is therefore critical that our system generates the cutout boundary with very little delay.

5.3 Framework

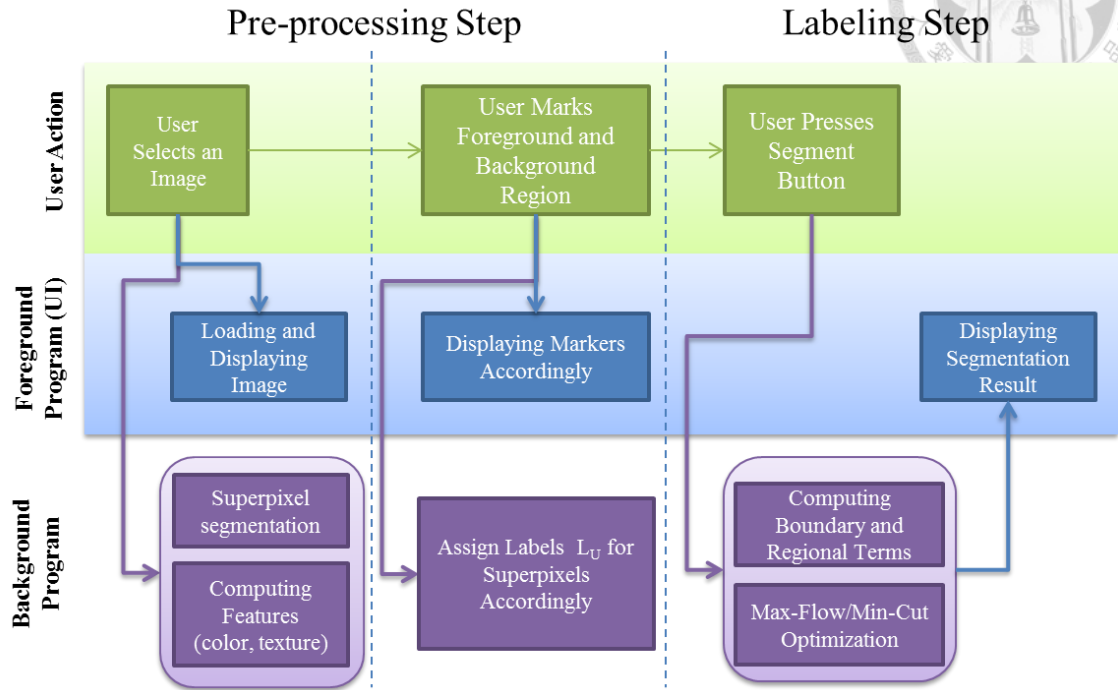


Fig. 5.2 Flowchart of our system.

Our complete flowchart includes user interaction, foreground program (user interface, i.e., UI), and background program three parts. In Fig. 5.2, the first row shows user interaction. User interaction triggers both foreground and background program to do the next step. The second row shows UI's reaction to user commands. The third row shows actions of background program, where lies our core algorithm.

User action triggers foreground program and background program at the same time. Background program and foreground program work independently. The time delay in Labeling Step is counted as the response time of our system. Although we expect the Pre-processing Step to be finished as soon as user done drawing scribbles, we add a check point before the Labeling Step to make sure we have all the materials ready for the final step.

5.4 Desired Properties of Superpixel Segmentation

For superpixel to be useful they must be fast, easy to use, and produce high quality segmentations. However, most state-of-the-art superpixel methods still suffer from high computational cost. Watersheds are widely used in image segmentation because there exist numerous and efficient algorithms that are easy to implement. However, segmentation results from watersheds may suffer from leaks and degeneracy of the solution on the plateaus of the weight function.

In our case, we look for superpixel methods having the following properties:

- Given superpixel number control: This property allows us to experiment our proposed framework on different sizes of superpixels.
- Produce superpixels of similar size: If superpixels have similar sizes, they would likely to be in more regular arrangement. For instance, every superpixel tends to have similar numbers of neighbors, which is close to the original arrangement of pixels. Thus, in max-flow/min-cut optimization, every superpixel will be “equally treated” when we consider the boundary penalty.
- Short processing time: As noted before, although we do not have to choose the fastest superpixel method, we still hope to restrict the processing time under one or two seconds, so that the process would be over before the user completes marking and starts to anticipate the result.

From the comparison results in 4.4 and 4.5, we find Simple Linear Iterative Clustering Superpixels (SLIC) and Energy Rate Superpixels (ERS) meet our requirements. They both produce compact and equal-sized superpixels that have good boundary adherence. SLIC is extremely fast. One downside is that the compactness factor affects the superpixel results tremendously, and is hard to tune the right parameter

for each image. Achanta et al. later published the zero-parameter version of SLIC, which is called SLICO. SLICO adaptively chooses the compactness parameter for each superpixel differently and relieves us from assigning the parameter by ourselves. SLICO works as fast as SLIC. The improvement hardly takes any toll on processing time. On the other hand, ERS is slower than SLIC (still faster than other recent methods); however, ERS has the best performance in the segmentation quality benchmark in 4.5. ERS also offers control over number of superpixels, which suits our need perfectly.

We only briefly explained the advantage of these two algorithms and why we use these two algorithms in our pre-segmentation step. For further information of the algorithms please refer to Appendix A.

5.5 Proposed Segmentation Algorithm

Here we describe the energy function and the max-flow/min-cut algorithm we use in our proposed framework.



5.5.1 The Regional Term

Our energy function is adapted from [5]. As we introduced earlier in 3.7, the regional term only considers the color information of the superpixel. We propose adding texture feature into regional term.

First, we define the color distance in the same fashion as [5], except that we do not do k-means clusters beforehand, because the number of superpixels in our framework is much smaller than in [5]. Also, we evaluate the color distance in CIELAB color space instead of RGB color space. The mean colors of the foreground and background segments are denoted as $\{K_n^F\}$ and $\{K_m^B\}$ respectively ($[l, a, b]$ three dimensions). We use l_i to represent a node (a superpixel) i labeled i . i could be F (foreground), B (background), or U (uncertain region). We define $l_i = 1$ if i belongs to F , and $l_i = 0$ if i belongs to B . Then, for each node i , we compute the minimum distance from its color $C(i)$ to foreground clusters as

$$dC_i^F = \min_n \|C(i) - K_n^F\| , \quad (5.1)$$

and to background clusters as

$$dC_i^B = \min_n \|C(i) - K_n^B\| . \quad (5.2)$$

Unlike color feature, a single pixel cannot provide any information about texture. On the other hand, when we consider a larger area, texture feature becomes meaningful. Since our proposed framework uses larger sizes of superpixels to speed up the graph cut

process, we incorporate texture feature into regional term.

We choose Log-Gabor Filter [56] to extract texture feature. Log-Gabor function is proposed by Field in 1987. Field suggests that log Gabor functions would be able to encode natural images more efficiently than ordinary Gabor functions, which over-represent the low frequency components and under-represent the high frequency components in any encoding. After evaluating the processing time and quality of the segmentation results, we choose Log-Gabor functions of scale = 2 and orientation = 4 as our texture feature (eight dimensions in total).

Similarly, for each node i , we compute the minimum distance from its mean texture $T(i)$ to foreground clusters as

$$dT_i^F = \min_n \|T(i) - T_n^F\|, \quad (5.3)$$

and to background clusters as

$$dT_i^B = \min_n \|T(i) - T_n^B\|. \quad (5.4)$$

Therefore, our regional term is defined as follows:

Table 5.1 Definitions of regional term.

	$l_i = 1$	$l_i = 0$
$i \in F$	0	∞
$i \in B$	∞	0
$i \in U$	$\frac{dC_i^F}{dC_i^F + dC_i^B} + \frac{dT_i^F}{dT_i^F + dT_i^B} \cdot \alpha$	$\frac{dC_i^B}{dC_i^F + dC_i^B} + \frac{dT_i^B}{dT_i^F + dT_i^B} \cdot \alpha$

where F and B denote foreground and background respectively, and $U = V \setminus \{F \cup B\}$ (the uncertain region).

5.5.2 The Boundary Term

We take the definition of boundary term from [5], but change the evaluation from RGB color space to CIELAB color space. B as a function of the color gradient between two nodes i and j :

$$B(i, j) = |l_i - l_j| \cdot g(C_{ij}) \quad (5.5)$$

where $g(\xi) = \frac{1}{\xi + 1}$, and $C_{ij} = \|C(i) - C(j)\|^2$ is the L2-Norm of the LAB color difference

5.5.3 Max-Flow/Min-Cut Algorithm

After computing boundary and regional term of every superpixel, we can use a max-flow/min-cut algorithm to solve the labeling problem and get our final segmentation result. We apply Boykov and Kolmogorov's max-flow/min-cut algorithm [4] in this step to do the graph cut segmentation¹¹.

¹¹ http://vision.csd.uwo.ca/wiki/vision/upload/d/d7/Bk_matlab.zip

5.6 Summary

In this section, we summarize the process of our background program in Table 5.2. and

Table 5.3.

Table 5.2 Algorithm of ur interactive segmentation framework

Algorithm The proposed segmentation framework (pre-processing step)	
Input:	Image I , number of superpixels K
Output:	Pre-segmented image S , feature look up table C and T
1:	Segment I with selected superpixel method and store the result in S
2:	Store color values of I in LAB space, each pixel has a color vector <i>lab</i> (3-dimensional)
3:	Apply Log-Gabor filter to I and store the response, each pixel has a texture vector <i>g</i> (8-dimensional)
4:	for each superpixel i do
5:	P = pixels in i
6:	$\mathbf{C}(i) = \sum_P lab / \#P$, $\mathbf{T}(i) = \sum_P g / \#P$
7:	end for

Table 5.3 Algorithm of ur interactive segmentation framework

Algorithm The proposed segmentation framework (labeling step)	
Input:	Pre-segmented image S , feature look up table C and T , label L_U marked by user, parameter λ , α
Output:	labels L
1:	for each superpixel i do
2:	if $i \in U$ then
3:	Compute color distance with C , T , L_U using (5.1), (5.2)
4:	Compute color distance with C , T , L_U using (5.3), (5.4)
5:	end if
6:	Assign boundary and regional term with Table 5.1 and (5.5)
7:	end for
8:	Max-Flow/Min-Cut Optimization : solve L by [4]



To quantitatively evaluate the accuracy of the segmentations produced by our algorithm, we applied it to the Microsoft GrabCut dataset [2]. As noted in [31], this dataset is not well suited to evaluating interactive scribble-based segmentation because it assumes the user loosely traces the contour of the desired object. As such, it provides far more seeds than are typically provided with interactive scribbles, and the seeds are more uniformly placed on either side of the boundary. We believe that interactive scribble-based methods cannot be evaluated with static seeds—once the user places the first scribble the resulting scribbles are dependent on the segmentation result from that and each successive one. However, to our knowledge, this is the only evaluation database to provide seeds, so we compute our results on this dataset for comparison. (Recently Santner et al. [40] published the first benchmark for interactive scribble based segmentation, but it targeted at multilabel segmentation, which is also not suitable for evaluating our algorithm.) The size of the images in the GrabCut dataset varies. There are mostly [481x231], [640x480] and [450x600].

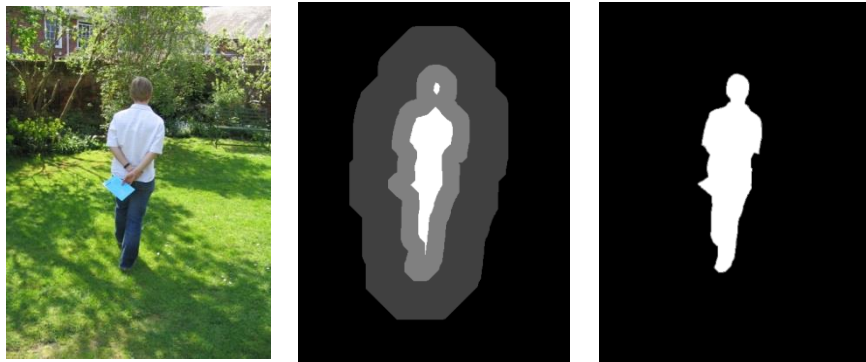


Fig. 6.1 Example of GrabCut dataset. Left column: original image. Middle column: the trimap image with user defined foreground (white) background (dark grey) and unknown (light grey). Right column: ground truth.

We evaluate results by Error Rate defined in [9]:

$$error\ rate = \frac{\#misclassified\ pixels}{\#unknown\ area\ pixels} \quad (5.6)$$



Table 6.1 reports the average Error Rate and Time Delay over 50 images. The statistical results are either provided by authors in their published works or tested by us using codes or executables provided by authors. To note that the methods chosen for comparison are all “scribbled based.” That means they have similar user interface and user interactions with us.

We can see that our average processing is close what we aim for, and outperforms all other methods. For Graph Cut segmentation, we choose Vicente et al.’s work for comparison. We obtained their Error Rate directly from their paper. Since they did not distribute source code, and response time not only depend on the algorithm but also on the implementation, we did not reimplement their algorithm. Instead, we argue that because the processing time of max-flow/min-cut optimization rises significantly with the number of nodes, the processing time of graph cut segmentation without pre-segmentation must be much higher than ours.

We list our Achievable Segmentation Error Rate (ASE) with the two different superpixel methods in the table. We use the algorithm in Table 6.2 to estimate the lowest error rate we could reach. We also put the results of our method using no texture information for comparison.

Additionally, we define

$$Node\ Ratio = \frac{\#pixel\ in\ original\ image}{\#superpixel\ after\ pre-segmentation} \quad (5.7)$$

We assume that Error Rate rises with Node Ratio. Visually the segmentation boarder tends to be coarser when the Node Ratio is high.

Table 6.1 Performance evaluation Microsoft GrabCut Database [3].

Segmentation Model	Error Rate(%)	Response Time
Li et al. (Lazy Snapping) [2004]	15.1	0.005s-1.129s, mean 0.339 ¹² (C++ with MATLAB wrapper)
Grady (Random Walker) [21]	10.9	0.15s - 0.83s, mean 0.47s (C++ with MATLAB wrapper)
Couprie et al. (Powerful Watershed) [25]	10.4	0.12s - 0.36s, mean 0.25s (C++)
Our Method without using texture information (ERS, Node Ratio = 100))	8.2	-
Our Method (ERS, Node Ratio = 100)	7.7	0.020 – 0.101s, Mean 0.058s (C++/MATLAB)
Graph Cut (Vicente et al.) [16]	6.7	-
ASE of our algorithm (SLICO, Node Ratio = 100)	6.0	-
Duchenne et al. (Segmentation by Transduction) [41]	5.4	2s – 3mins (C++)
ASE of our algorithm (ERS, Node Ratio = 100)	4.9	-
Price et al. Geodesic Graph Cut [42]	4.8	0.2s – 2.6s (C++)

¹² We only timed the max-flow/min-cut optimization step since we were not using the source code provided by the authors and the processing time greatly depends on implementation. We show that under the same optimization scheme, our response time is already shorter than their processing time of max-flow/min-cut algorithm. For more details about their response time please refer to [5].

Next, we show visual comparison of our result and three other methods' results in Fig. 6.2 and Fig. 6.3. Duchenne et al.'s [41] result were obtained from their conference paper. Grady's [21] and Couprie et al.'s [25] results were generated using source codes provided by the authors. In addition, we implemented Li et al.'s [5] method by ourselves. Since we might implement their algorithm differently, we only show the processing time of the max-flow/min-cut optimization step. Just to show that using the same optimization scheme they have longer processing time than us theoretically and practically. In Fig. 6.4 - Fig. 6.7, we show more segmentation results for visual comparison

In Fig. 6.2, we can see that our result is clearly better than Grady and Couprie et al.'s. Our algorithm is able to overcome the typical shrinking bias issues as illustrated in 3.5.1. Although our algorithm cuts out most parts of the person, our segmentation boarder around the hair and legs of the person is not as smooth and accurate as Duchenne et al.'s result. Part of the reason is that the boundary of superpixel does not exactly fit the boundary of the object. With ERS pre-segmentation and Node Ratio = 100, the ASA of this test image is 4.14%, while our result's error rate is 6.63%. Fig. 6.3 shows one of the hard cases in the dataset. The object has similar color to the background. To note that our algorithm segment out the left leg of the subject without confusing with the background. Our result outperforms Grady's and Couprie et al.'s result and is comparable to Duchenne et al.'s.

Table 6.2 Algorithms of Calculating ASE

Algorithm	Calculate ASE
Input:	Image I
Output:	Error Rate e
1:	Segment I with selected superpixel method and store the result in S
2:	for each superpixel i in S
3:	if the area of superpixel overlapped with groundtruth > 50%
4:	label the superpixel to foreground
5:	end if
6:	end for
7:	Calculate e of the result labels with (5.6)





(a) Input image



(b) Trimap



(c) ASE with ERS,
Node Ratio = 100



(d) ASE with SLICO,
Node Ratio = 100



(e) Duchenne et al. [41]



(f) Couprie et al. [25]



(g) Grady [21]



(h) Li et al. [5]



(h) Proposed Method

Fig. 6.2 Visual comparison on dataset [2].



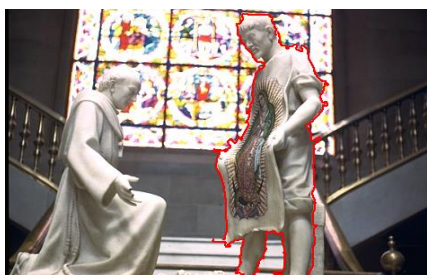
(a) Input image



(b) Trimap



(c) ASE with SLICO,
Node Ratio = 100



(d) ASE with ERS,
Node Ratio = 100



(e) Duchenne et al. [41]



(f) Grady[21]



(g) Couprie et al. [25]

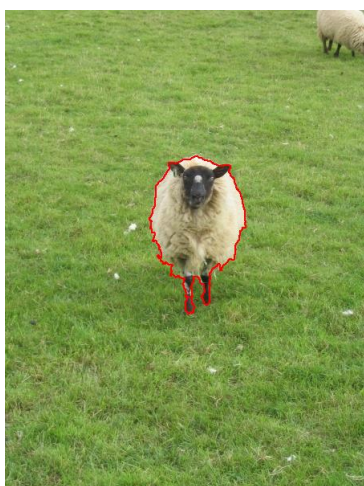


(h) Li et al. [5]



(i) Proposed Method

Fig. 6.3 Visual comparison on dataset [2]. The error rate of (c)-(h) are 5.19%, 4.39%, 9.15%, 15.06%, 14.20% and 9.42% respectively.



(a) Proposed Method



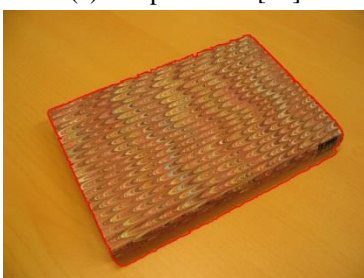
(b) Grady[21]



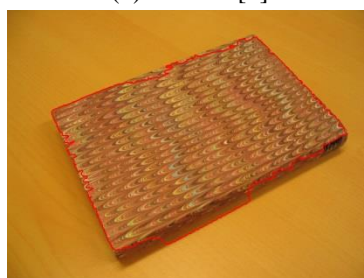
(c) Couprie et al. [25]



(d) Li et al. [5]



(e) Proposed Method



(f) Grady[21]

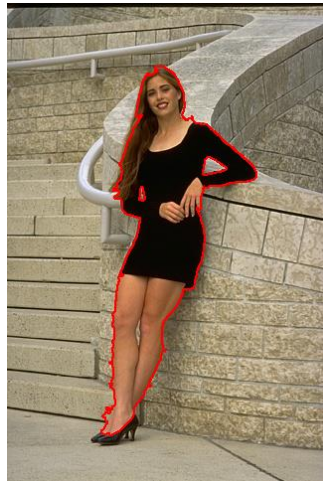


(g) Couprie et al. [25]



(h) Li et al. [5]

Fig. 6.4 Visual comparison on dataset [2].



(a) Proposed Method



(b) Grady[21]



(c) Couprie et al. [25]



(d) Li et al. [5]



(e) Proposed Method



(f) Grady[21]



(g) Couprie et al. [25]



(h) Li et al. [5]

Fig. 6.5 Visual comparison on dataset [2].



(a) Proposed Method



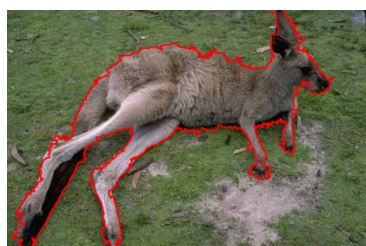
(b) Grady[21]



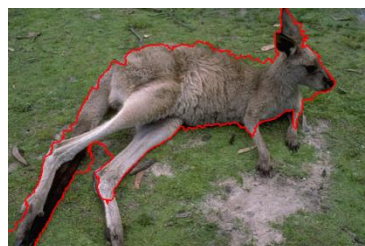
(c) Couprie et al. [25]



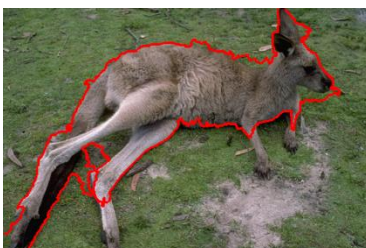
(d) Li et al. [5]



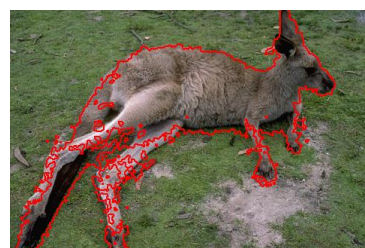
(e) Proposed Method



(f) Grady[21]



(g) Couprie et al. [25]



(h) Li et al. [5]

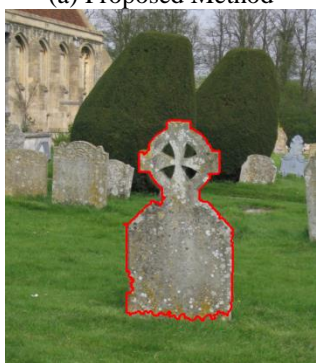
Fig. 6.6 Visual comparison on dataset [2].



(a) Proposed Method



(b) Grady[21]



(c) Couprie et al. [25]



(d) Li et al. [5]



(e) Proposed Method



(f) Grady[21]



(g) Couprie et al. [25]



(h) Li et al. [5]

Fig. 6.7 Visual comparison on dataset [2].

6.2 Discussion

With superpixel pre-segmentation, we shorten the response time of our program. However, we have to sacrifice some quality at the same time. In this section, we show statistic results of time delay and error rates regarding segmentation results using different superpixel methods with different node ratio. We would discuss the best strategy according to the comparison.

6.2.1 Processing Time

Fig. 6.8 - Fig. 6.10 are the scatter charts of the processing time of our algorithm with ERS pre-segmentation. We can see that the response time is generally related to the number of nodes (number of superpixels). Apart from this factor, the processing time of max-flow/min-cut optimization may also be effected by the number of “edges” in the flow network, i.e., the number of neighbors of each superpixel. Thus, images with the same number of superpixels do not necessarily have the same max-flow/min-cut processing time. On the other hand, we may also spend different processing time on computing energy terms for the same number of superpixels. Recall that in 5.5.1, we use different methods to assign regional values for superpixels with different labels(U, B or F). We need to do some calculation which consists of massive loops before assigning values to superpixels $\in U$. Thus, different percentage of the three labels results in different processing time. Thus, we cannot control the exact processing time of this part by fixing the number of superpixels. However, overall, more superpixels leads to more processing time.

In Fig. 6.11, we show the response time of all 50 images with different node ratio. We can see that almost all 50 images have response time $< 100\text{ms}$ with node ratio = 100,

which is the parameter we use to generate all the visual results in this thesis. If we set node ratio = 200, response time can be shorten down to under 40ms. In Fig. 6.12 - Fig. 6.15, we show similar results of our algorithm with SLICO pre-segmentation.

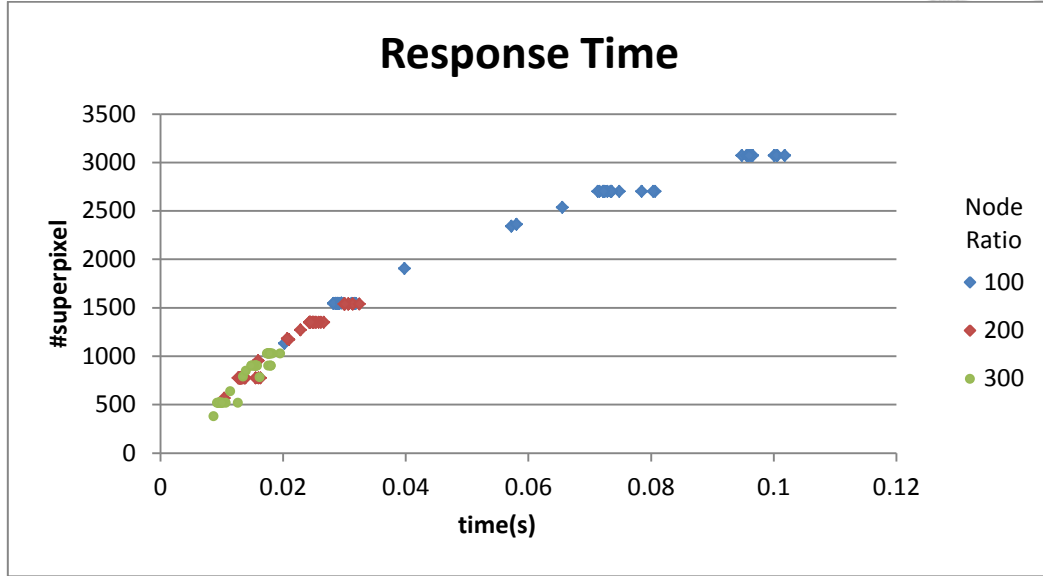


Fig. 6.8 Response time with respect to different number of ERS superpixels.

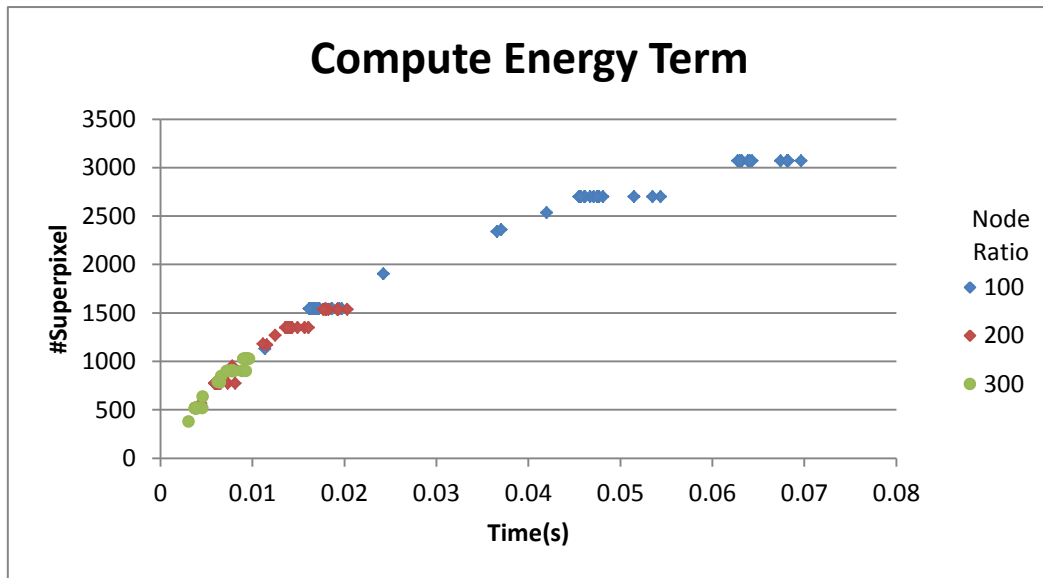


Fig. 6.9 Processing time of computing energy term with respect to different number of ERS superpixels.

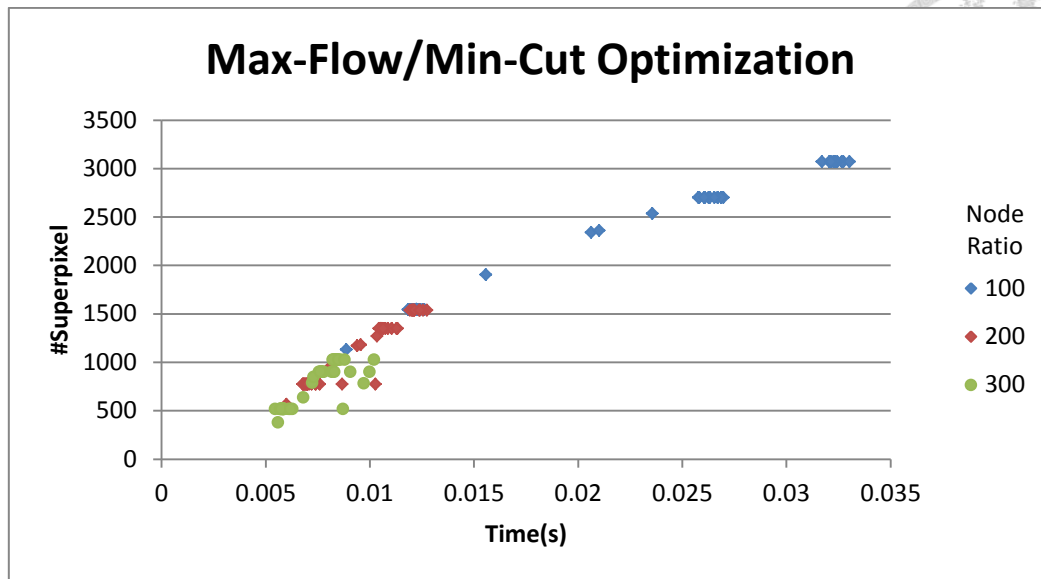


Fig. 6.10 Processing time of with respect to different number of ERS superpixels.

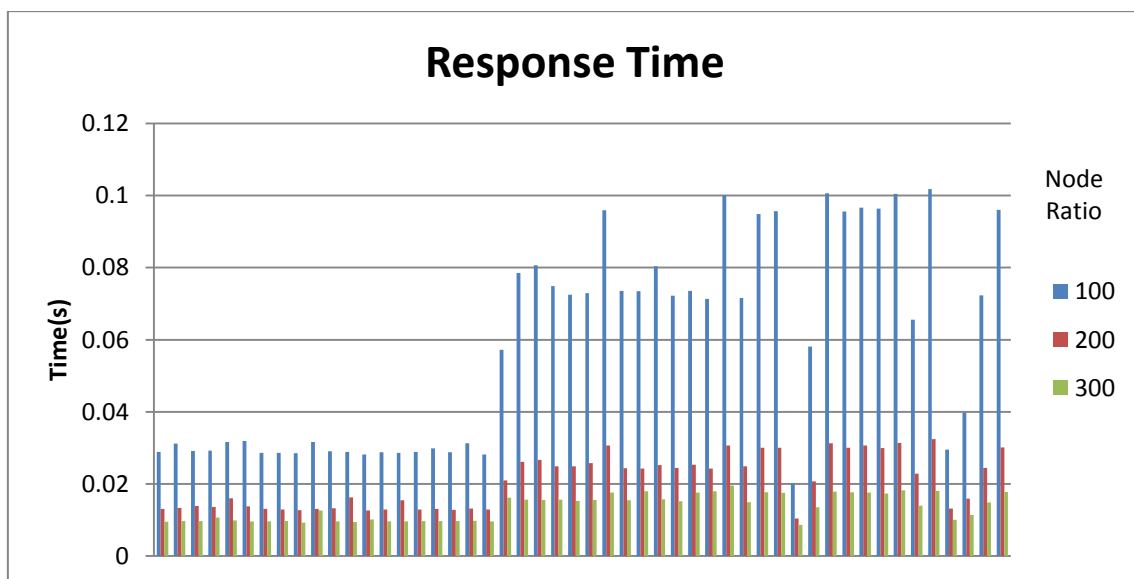


Fig. 6.11 Response time of 50 images with different node ratio. (ERS)

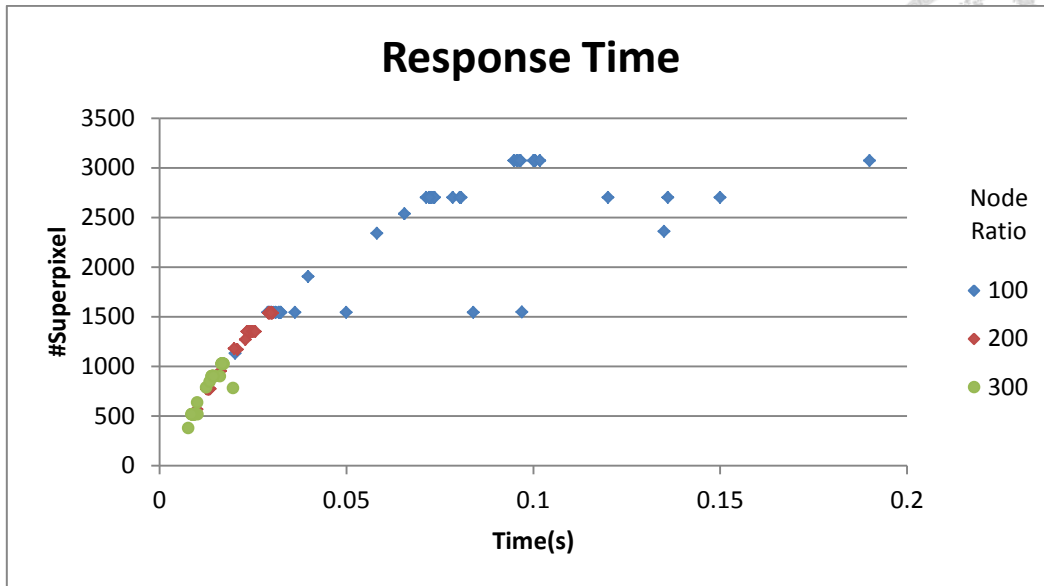


Fig. 6.12 Response time with respect to different number of SLICO superpixels.

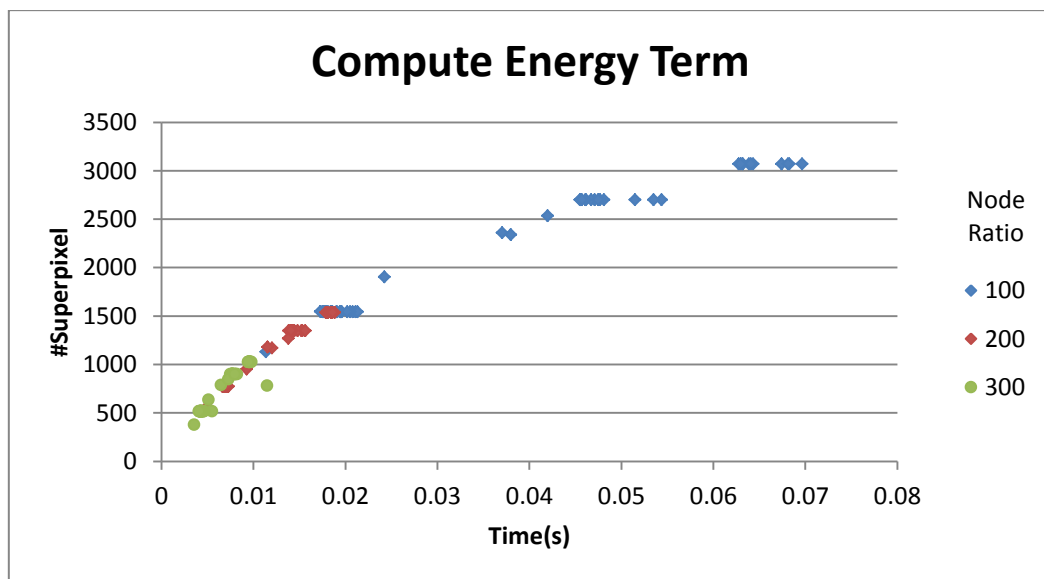


Fig. 6.13 Processing time of computing energy term with respect to different number of SLICO superpixels.

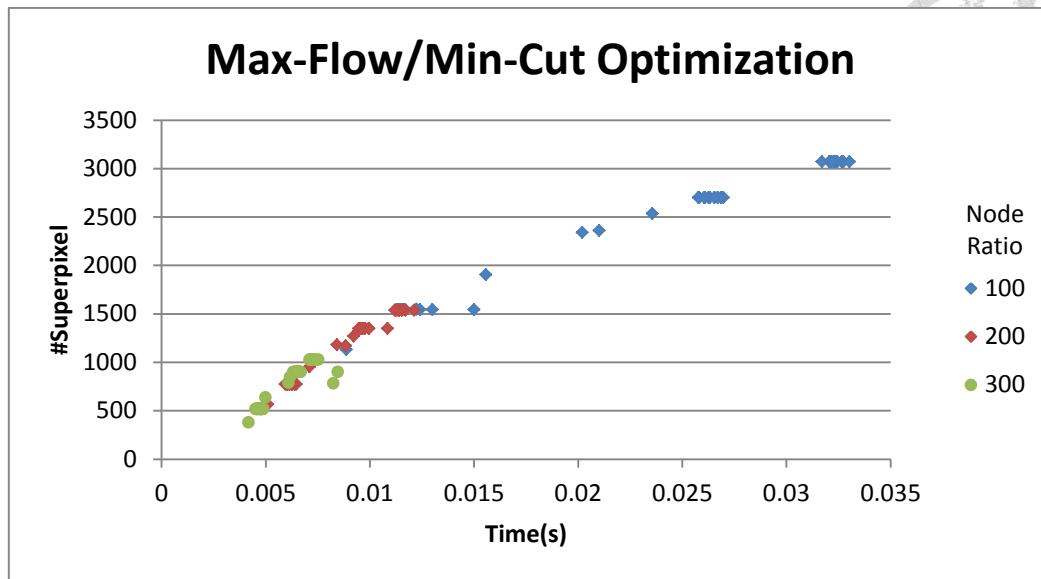


Fig. 6.14 Processing time of with respect to different number of SLICO superpixels.

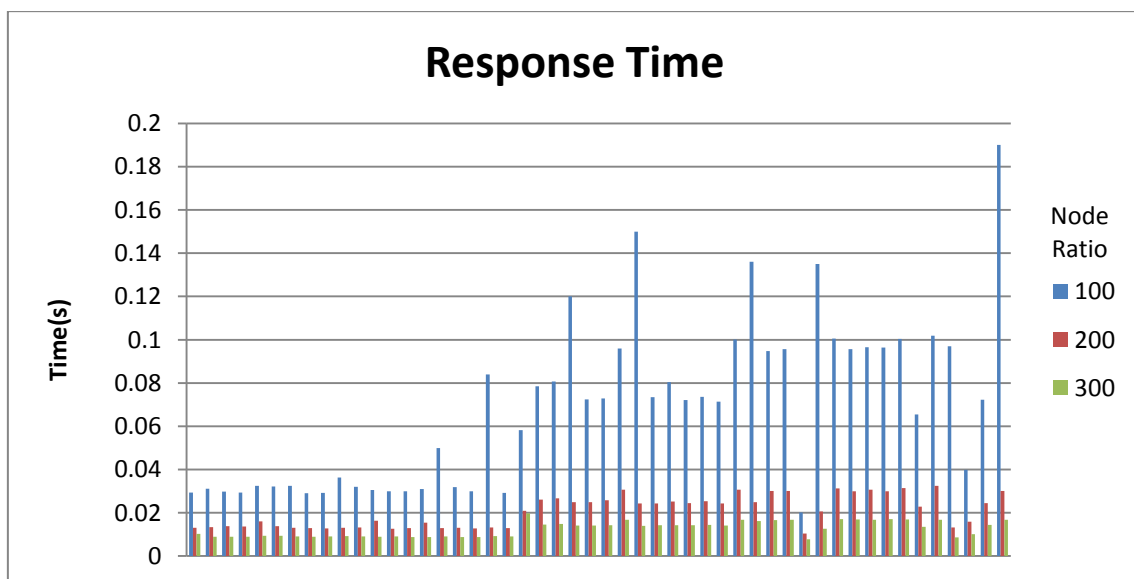


Fig. 6.15 Response time of 50 images with different node ratio. (SLICO)

Additionally, to verify our assumption that preprocessing time is less than the time user needed for marking object, we provide statistical results in Table 6.3. For SLICO pre-segmentation, the total processing time is around 1.5 seconds. For ERS pre-segmentation, the total processing time is about 2.5 seconds, which we consider reasonable.

Table 6.3 Preprocessing time of two superpixel methods (in second).

Process	Node Ratio	100	200	300
SLICO	Pre-segmentation	0.291	0.375	0.457
	Computing Features	1.249	1.043	0.856
	Preprocessing	1.540	1.418	1.313
ERS	Pre-segmentation	1.392	1.414	1.512
	Computing Features	1.117	0.817	0.717
	Preprocessing	2.509	2.231	1.289

6.2.2 Achievable Segmentation Error

Here we list our error rate and ASE of different node ratio and superpixel algorithms. As the node ratio rises, ASE also rises. This tendency may be caused by the drop of boundary recall of superpixel segmentation.

Table 6.4 Error rate of two superpixel methods and various node ratio.

Algorithm	Node Ratio	100	200	300
ERS	ASE	4.9387	6.0141	6.6641
	Proposed Method	7.7543	8.1311	8.3464
SLICO	ASE	6.0416	6.8367	7.8556
	Proposed Method	8.4763	9.1628	9.9610

6.2.3 Graph Cut Segmentation Issues Revisited

1. Shrinking Bias

From Fig. 6.2 to Fig. 6.7, we can see that our algorithm is able to avoid boundary short cutting. However, in some case, the shrinking bias still exists and results in high error rate. In Fig. 6.16, our algorithm fails to segment out the right arm of the boy. The colors spread out the image are highly similar, so adding texture information to provide greater region coherence does not work. This issue needs further work to be resolved.

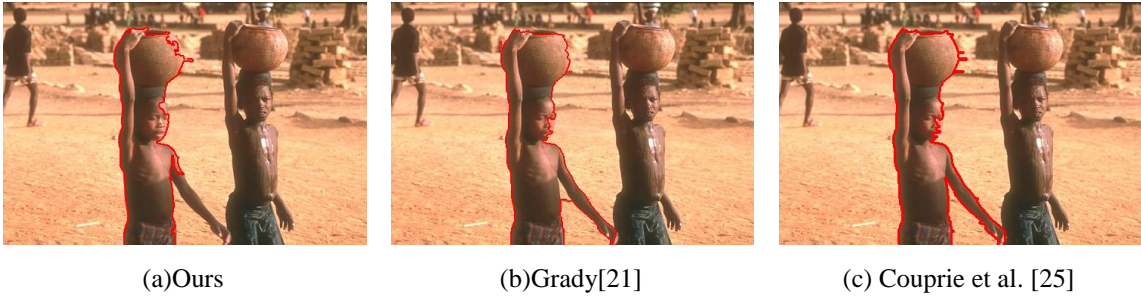


Fig. 6.16 Visual comparison of our failure case.

2. Efficiency

Table 6.1 shows that we have significantly improved the response time with our framework and state-of-the-art superpixel methods.

3. Vulnerability to Noise

To study the robustness of our algorithm against image noise, we add Gaussian noise to images of dataset [3] and then rerun the segmentation algorithms on noisy images. For our result, we run our program with the same parameters as in Table 6.1. For Random Walker[21]¹³ and Power Watershed[25]¹⁴, we use the source code from

¹³ <http://powerwatershed.sourceforge.net/>

¹⁴ <http://cns.bu.edu/~lgrady/software.html>

their project page with their default settings. We can see from Table 6.5 that our error rate rises slower than two other methods.

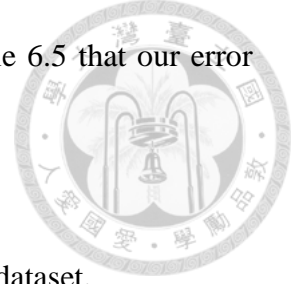


Table 6.5 Error Rate of segmentation results on noisy dataset.

	Original	Add Gaussian noise with mean=0, variance = 0.001	Add Gaussian noise with mean=0, variance = 0.01
Proposed Method	7.7	7.9	8.6
Grady [21]	10.9	11.3	12.3
Couprie et al. [25]	10.4	11.5	12.5

6.3 Summary

In this chapter, we give discussion on processing time and error rate of our algorithm regarding two different superpixel methods. Overall, ERS can achieve lower error rate because of its high boundary recall rate. However, if we deal with images of larger size, the pre-processing time may be longer than the interaction time the user needs, and thus the user would perceive an obvious delay. On the other hand, SLICO has stable and short segmentation time, so it has better ability to process larger images on a real-time interactive system.

Chapter 7 Conclusion and Future Work



7.1 Conclusion

In this thesis, we provide an overview of interactive segmentation methods. In addition, we introduce algorithms of Graph Cut Segmentation in details. We also discuss typical issues in graph cut segmentation and prior improvements trying to tackle them. Then, we reviewed superpixel methods in two categories and give a thorough comparison on Neubert and Protzel's benchmark [43].

We propose an interactive segmentation framework aims at making the system real-time. We incorporate texture information into the popular graph cut optimization framework to further lower the segmentation error rate. Quantitative evaluation is made on GrabCut dataset [2]. Our framework is faster than state-of-the-art methods. While sacrificing quality for shorter response time, our segmentation results still beat some of the recent methods. Additionally, our method relieves some typical issues in graph cut segmentation. Besides improving the efficiency, our algorithm is proven to be more robust than some recent methods when processing noisy images.

7.2 Future Work

We could further speed up the program by implementing the code fully by C++. (We are now half C++ half MATLAB.) Additionally, since we apply the same calculation to all superpixels, the process can be implemented in parallel computing.

We have not resolved the shrinking bias problem caused by the nature of graph cut segmentation. One way to solve this problem is to adaptively tuning the combination of different feature information when assigning the energy terms. For example, when the image's foreground and background color are similar, greater weight is given to the texture (or other feature) component to provide greater region coherence and avoid boundary short cutting.

Appendix A



In this appendix, we introduce the algorithm of Simple Linear Iterative Clustering Superpixel (SLIC) in details. In addition, we introduce the SLICO algorithm, which is the zero-parameter version of SLIC. In our final implementation, we choose SLICO over SLIC. We also give a brief comparison between SLIC and SLICO.

A.1 Simple Linear Iterative Clustering Superpixel (SLIC)

As introduced in 4.2, SLIC is fast, memory efficient, and also exhibits state-of-the-art boundary adherence. It is an adaptation of k -means for superpixels generation, with two important distinctions:

1. The number of distance calculations in the optimization is reduced by limiting the search space to a region proportional to superpixel size. This reduces the complexity to be linear in the number of pixels N and independent of the number of superpixels K .
2. A weighted distance measure combines color and spatial proximity while simultaneously providing control over the size and compactness of the superpixels.

SLIC is simple to use and understand. By default, the only parameter of the algorithm is K , the desired number of approximately equally sized superpixels. For a color image in the CIELAB color space, the clustering procedure begins with an initialization step where K initial cluster centers $C_k = [l_k \ a_k \ b_k \ x_k \ y_k]^T$ with $k = [1, K]$ are sampled on a regular grid spaced S pixels apart. To produce roughly equally sized superpixels, the grid interval is $S = \sqrt{N/K}$, so the approximate size of each superpixel is therefore N/K pixels for an image with N pixels. The centers are moved to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood. This is done to avoid centering a superpixel on an edge and to reduce the chance of seeding a

superpixel with a noisy pixel.

Next, in the assignment step, each pixel i is associated with the nearest cluster center whose search region overlaps its location, as depicted in Fig. 2.4. This is the key to speeding up our algorithm because limiting the size of the search region significantly reduces the number of distance calculations, and results in a significant speed advantage over conventional k -means clustering where each pixel must be compared with all cluster centers. Since the expected spatial extent of a superpixel is a region of approximate size $S \times S$, the search for similar pixels is done in a region $2S \times 2S$ around the superpixel center.

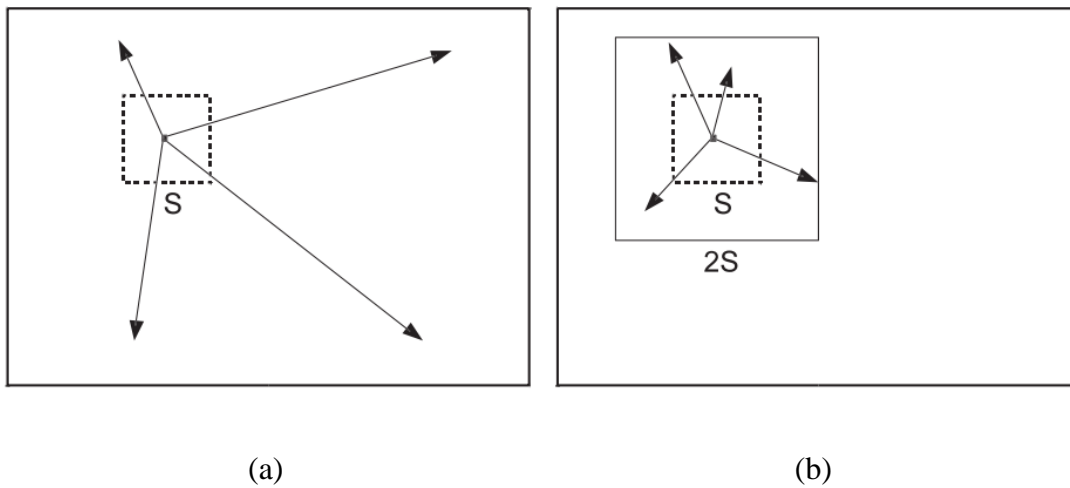


Fig. A.1 Reducing the superpixel search regions. (a) standard k -means searches the entire image. (b) SLIC searches a limit region.

There is a problem that how to define the distance measure D . While the maximum possible distance between two colors in the CIELAB space is limited, the spatial distance in the xy plane depends on the image size. It is not possible to simply use the Euclidean distance in this 5D space without normalizing the spatial distances. In order

to cluster pixels in this 5D space, therefore a new distance measure that considers superpixel size is introduced. Using it enforce color similarity as well as pixel proximity in this 5D space such that the expected cluster sizes and their spatial extent are approximately equal. The measure is defined by combining the color proximity and spatial proximity normalized by their respective maximum distances within a cluster, N_c and N_s , as follows:

$$d_c = \sqrt{(l_j - l_i)^2 + (a_j - a_i)^2 + (b_j - b_i)^2} \quad (\text{A.1})$$

$$d_s = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \quad (\text{A.2})$$

$$D' = \sqrt{\left(\frac{d_c}{N_c}\right)^2 + \left(\frac{d_s}{N_s}\right)^2} = \sqrt{\left(\frac{d_c}{m}\right)^2 + \left(\frac{d_s}{S}\right)^2} \quad (\text{A.3})$$

where d_c and d_s are distances in color and spatial space, respectively. The parameter m is a constant to represent the respective maximum color distance N_c , and the maximum spatial distance expected within a given cluster should correspond to the sampling interval, therefore $N_s = S$.

In SLICO, the distance measure D is defined as

$$D = \sqrt{d_c^2 + \left(\frac{d_s}{S}\right)^2 m^2}. \quad (\text{A.4})$$

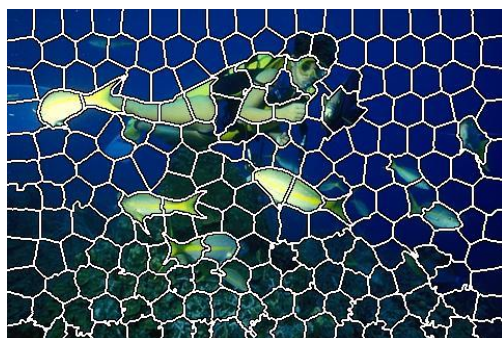
By defining D in this manner, m can be adjusted to weigh the relative importance between color similarity and spatial proximity. When m is large, spatial proximity is more important and the resulting superpixels are more compact (i.e., they have a lower area to perimeter ratio). When m is small, the resulting superpixels adhere more tightly to image boundaries, but have less regular size and shape. When using the CIELAB color space, m can be in the range [1,40].

Once each pixel has been associated to the nearest cluster center, an update step

adjusts the cluster centers to be the mean $[l\ a\ b\ x\ y]^T$ vector of all the pixels belonging to the cluster. The L_2 norm is used to compute a residual error E between the new cluster center locations and previous cluster center locations. The assignment and update steps can be repeated iteratively until the error converges, but in most of time that 10 iterations suffices for most images, and report all results in this paper using this criteria.

Finally, a post-processing step enforces connectivity by reassigning disjoint pixels to nearby largest superpixels. We show the complete algorithm in Table A.1

In the figure below, the first column of images shows SLIC output with a constant compactness factor for all superpixels, while the second column of images shows the output of SLICO, which chooses the compactness factor adaptively for each superpixel. If the image is smooth in certain regions but highly textured in others, SLIC produces smooth regular-sized superpixels in the smooth regions and highly irregular superpixels in the textured regions. Thus, it becomes tricky choosing the right parameter for each image. On the other hand, in SLICO, the user no longer has to set the compactness parameter or try different values of it. SLICO adaptively chooses the compactness parameter for each superpixel differently. This generates regular shaped superpixels in both textured and non-textured regions alike. To note that the improvement comes with hardly any compromise on the computational efficiency - SLICO continues to be as fast as SLIC.



SLIC

SLICO

Fig. A.2 Segmentation results of SLIC and SLICO.

Table A.1 Algorithm of SLICO superpixel segmentation.

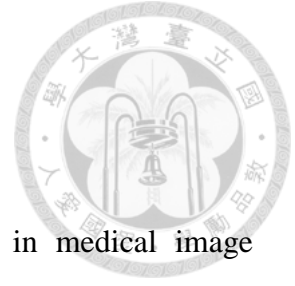
Algorithm SLICO Superpixel Segmentation

Input: Image with N pixels, number of superpixels K , compactness parameter m

Output: Segmented map


- 1: Initialize K cluster centers $C_k = [l_k \ a_k \ b_k \ x_k \ y_k]^T$ by sampling pixels at regular grid step S , $S = \sqrt{N/K}$.
 - 2: Move cluster centers to the lowest gradient position in a 3×3 neighborhood.
 - 3: Set label of pixel i , $l(i) = -1$ for each pixel.
 - 4: Set distance between nearest cluster center and pixel i , $d(i) = \infty$ for each pixel.
 - 5: **repeat**
 - 6: **for** each cluster center C_k **do**
 - 7: **for** each pixel i in a $2S \times 2S$ region around C_k **do**
 - 8: Compute the distance D between C_k and i with (A.4)
 - 9: **if** $D < d(i)$ **then**
 - 10: Set distance between nearest cluster center and pixel i , $d(i) = D$
 - 11: Set label of pixel i , $l(i) = k$
 - 12: **end if**
 - 13: **end for**
 - 14: **end for**
 - 15: Compute new cluster centers using the mean value of pixels in each cluster.
 - 16: Compute residual error E , the L_2 distance between previous centers and recomputed centers.
 - 17: **until** $E \leq \text{threshold}$
 - 18: Enforce connectivity by reassigning disjoint pixels.
-

REFERENCE

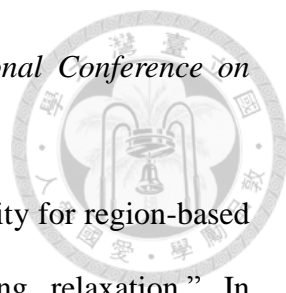


A. Interactive Image Segmentation

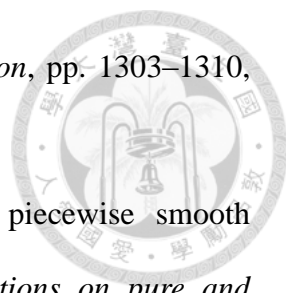
- [1] D. L. Pham, C. Xu, and J. L. Prince, "Current methods in medical image segmentation 1", *Annual Review of Biomedical Engineering*, vol. 2, no. 1, pp. 315-337, 2000.
- [2] C. Rother, V. Kolmogorov, and A. Blake. "Grabcut: Interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 309–314, 2004.
- [3] Y. Y. Boykov, and M. P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images," In *Proceedings of the IEEE International Conference on Computer Vision*, vol.1, pp.105-112, 2001.
- [4] Y. Boykov and V. Kolmogorov, "An experimental comparison of mincut/max-flow algorithms for energy minimization in vision." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, pp. 1124–1137, 2004.
- [5] Y. Li, J. Sun, C. K. Tang, and H. Y. Shum. "Lazy snapping," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 303-308, 2004.
- [6] F. R. Chung, "Spectral Graph Theory," vol. 92, American Mathematical Soc., 1997.
- [7] N. Biggs, "Algebraic Graph Theory", Cambridge University Press, 1974.
- [8] D. Greig, B. Porteous, and A. Seheult. "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society, Series B*, pp. 271–279, 1989.
- [9] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr, "Interactive Image

- 
- Segmentation Using an Adaptive GMMRF Model,” In *Proceedings of the European Conference on Computer Vision*, pp. 428-441, 2004.
- [10] M. Gleicher, “Image snapping,” In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 183-190, 1995.
- [11] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *International journal of computer vision*, vol. 1, no. 4, pp. 321-331, 1987.
- [12] H. Lombaert, Y. Sun, L. Grady, and C. Xu, “A Multilevel Banded Graph Cuts Method for Fast Image Segmentation,” In *Proceedings of the IEEE International Conference on Computer Vision*, vol. 1, pp. 259-265, 2005.
- [13] E. N. Mortensen and W. A. Barrett, “Intelligent scissors for image composition,” In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 191–198, 1995.
- [14] J. Wang, M. Agrawala, and M. F. Cohen, “Soft scissors: an interactive tool for realtime high quality matting,” *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 9:1-9:6, 2007.
- [15] V. Kolmogorov and Y. Boykov, “What metrics can be approximated by geo-cuts, or global optimization of length/area and flux,” In *Proceedings of the IEEE International Conference on Computer Vision*, vol. 1, pp. 564–571, 2005.
- [16] S. Vicente, V. Kolmogorov, and C. Rother, “Graph cut based image segmentation with connectivity priors,” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [17] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888-905, 2000.
- [18] V. Kolmogorov, Y. Boykov, and C. Rother, “Applications of parametric maxflow

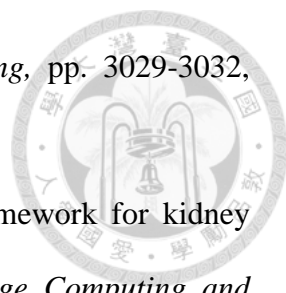
in computer vision,” In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1-8, 2007.

- 
- [19] T. Schoenemann, F. Kahl, and D. Cremers, “Curvature regularity for region-based image segmentation and inpainting: A linear programming relaxation,” In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 17-23, 2009.
- [20] A. K. Sinop and L. Grady, “A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm,” In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1-8, 2007.
- [21] L. Grady, “Random walks for image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1768–1783, 2006.
- [22] X. Bai and G. Sapiro, “A geodesic framework for fast interactive image and video segmentation and matting,” In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1–8, 2007.
- [23] A. Protiere and G. Sapiro, “Interactive image segmentation via adaptive weighted distances,” *IEEE Transactions on Image Processing*, vol. 16, no. 4, pp. 1046–1057, 2007.
- [24] L. J. Reese and W. A. Barrett, “Image editing with intelligent paint,” In *Proceedings of Eurographics*, vol. 21, no. 3, pp. 714–724, 2002.
- [25] C. Couprie, L. Grady, L. Najman, and H. Talbot, “Power watersheds: A new image segmentation framework extending graph cuts, random walker and optimal spanning forest,” In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 731-738, 2009.
- [26] D. Singaraju, L. Grady, and R. Vidal, “P-Brush: Continuous valued MRFs with normed pairwise distributions for image segmentation,” In *Proceedings of the*


IEEE Conference on Computer Vision and Pattern Recognition, pp. 1303–1310, 2009.

- 
- [27] D. Mumford, and J. Shah, “Optimal approximations by piecewise smooth functions and associated variational problems.” *Communications on pure and applied mathematics*, vol. 42, no. 5, pp. 577-685, 1989.
- [28] S. Geman, and D. Geman, “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721-741, 1984.
- [29] E. N. Mortensen, and W. A. Barrett, “Interactive segmentation with intelligent scissors.” *Graphical models and image processing*, vol. 60, no. 5, pp. 349-384, 1998.
- [30] L. D. Cohen, and R. Kimmel, “Global minimum for active contour models: A minimal path approach.” *International journal of computer vision*, vol. 24, no. 1, pp. 57-78, 1997.
- [31] L. Grady, “Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3D with application to segmentation.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 69-78, 2006.
- [32] D. Freedman and T. Zhang, “Interactive graph cut based segmentation with shape priors.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 755-762, 2005.
- [33] N. Vu and B. S. Manjunath, “Shape prior segmentation of multiple objects with graph cuts.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [34] H. Wang and H. Zhang, “Adaptive shape prior in graph cut segmentation.”

In *Proceedings of the IEEE Conference on Image Processing*, pp. 3029-3032, 2010.

- 
- [35] A. M. Ali, A. A. Farag and A. S. El-Baz, “Graph cuts framework for kidney segmentation with prior shape constraints.” In *Medical Image Computing and Computer-Assisted Intervention*, pp. 384-392, 2007.
- [36] O. Veksler, “Star shape prior for graph-cut image segmentation.” In *Proceedings of the IEEE Conference on Computer Vision*, pp. 454-467, 2008.
- [37] L. Vincent and P. Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583-598, 1991.
- [38] M. Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2097-2104, 2011.
- [39] O. Veksler and Y. Boykov, “Superpixels and supervoxels in an energy optimization framework,” In *Proceedings of the European Conference on Computer Vision*, pp. 211-224, 2010.
- [40] J. Santner, T. Pock, and H. Bischof, “Interactive multi-label segmentation,” Springer Berlin Heidelberg, pp. 397-410, 2011.
- [41] O. Duchenne, J. Y. Audibert, R. Keriven, J. Ponce, and F. Ségonne, “Segmentation by transduction.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1-8, 2008.
- [42] B. L. Price, B. Morse, and S. Cohen, “Geodesic graph cut for interactive image segmentation.” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3161-3168, 2010.

B. Superpixel

- 
- [43] P. Neubert and P. Protzel, “Superpixel benchmark and comparison,” In *Proceedings of Forum Bildverarbeitung*, 2012.
- [44] L. Vincent and P. Soille, “Watersheds in digital spaces: An efficient algorithm based on immersion simulations.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 1991.
- [45] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
- [46] A. Vedaldi and S. Soatto, “Quick shift and kernel methods for mode seeking,” In *Proceedings of the European Conference on Computer Vision*, pp. 705–718, 2008.
- [47] A. Levinstein, A. Stere, K. Kutulakos, D. Fleet, S. Dickinson, and K. Siddiqi. “Turbopixels: Fast superpixels using geometric flows,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no.12, pp. 2290–2297, 2009.
- [48] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, “SLIC superpixels compared to state-of-the-art superpixel methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.
- [49] J. Shi and J. Malik, “Normalized cuts and image segmentation.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.
- [50] T. Cour, F. Benezit, and J. Shi, “Spectral segmentation with multiscale graph decomposition,” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1124–1131, 2005.
- [51] P. Felzenszwalb and D. Huttenlocher, “Efficient graph-based image segmentation.”

International Journal of Computer Vision, vol. 59, no. 2, pp. 167–181, 2004.

- [52] Y. M. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, “Entropy rate superpixel segmentation,” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2097-2104, 2011.

C. Theorems and Methmetics

- [53] V. Vineet, and P. J. Narayanan, “CUDA cuts: Fast graph cuts on the GPU,” In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-8, 2008.
- [54] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898-916, 2011.
- [55] R. B. Miller, “Response time in man-computer conversational transactions,” In *Proceedings of the fall joint computer conference, ACM, part I*, pp. 267-277, 1968.
- [56] D. J. Field, “Relations between the statistics of natural images and the response properties of cortical cells,” *Journal of the Optical Society of America A*, vol. 4, no. 12, pp. 2379-2394, 1987.