Graph Cuts for Image Segmentation

Ph.D. Seminar Report

Submitted in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

by

Meghshyam G. Prasad
Roll No: 124058001

under the guidance of

Prof. Sharat Chandran

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Contents

Abstract

In computer vision, segmentation is the process of partitioning digital image into multiple regions (sets of pixels), according to some homogeneity criterion. The problem of segmentation is a well-studied one in literature and there are a wide variety of approaches that are used. Graph cuts has emerged as a preferred method to solve a class of energy minimization problems such as Image Segmentation in computer vision. Boykov et.al[3] have posed Image Segmentation problem as Energy Minimization in Markov Random Field and found approximately minimum solution using Graph cuts.

Min-Cut/Max flow algorithms for Graph cuts include both push-relabel methods as well as augmenting paths methods. Boykov and Kolmogorov [2] have developed an efficient method for finding augmenting path. Though experimental comparison shows this algorithm efficient over other, worst case complexity of it is very high. In [1], Voronoi based Push Preflow method is suggested to find min-cut, which not only exploits the structural properties inherent in image based grid graphs but also combines the basic paradigms of max-flow theory in a novel way.

Though Min-cut/Max-Flow based Graph cut methods can efficiently find partitions, those (partitions) may not be the desired ones. So, [7] have developed concept of Normalized cuts. Normalized cuts considers association within a cluster as well as the disassociation among clusters.

1 Introduction

The goal of image segmentation is to cluster pixels into salient image regions, i.e., regions corresponding to individual surfaces, objects, or natural parts of objects. It can identify the regions of interest in a scene or annotate the data. Each of the pixels in a region are 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 useful in many applications such as Medical Imaging(Tumor Detection), Face Recognition, Machine Vision etc.

1.1 Previous Approaches

As Image Segmentation problem is a well-studied in literature, there are many approaches to solve it. Detail review of various segmentation techniques is done in [5][6]. Most significant contributions are as follows:

1.1.1 Region Based Segmentation

A region-based method usually proceeds as follows:

- The image is partitioned into connected regions by grouping neighboring pixels of similar intensity levels.
- Adjacent regions are then merged under some criterion such as homogeneity or sharpness of region boundaries.
- Overstringent criteria create fragmentation; lenient ones overlook blurred boundaries and overmerge.

1.1.2 Watershed Segmentation

This method visualizes images in 3-dimensions: x, y, and intensity. In this technique we find watershed lines: Points at which a drop of water will be equally likely to fall to more than one minimum Instead of working on an image itself, this technique is often applied on its gradient image. So, each object is distinguished from the background by its up-lifted edges.

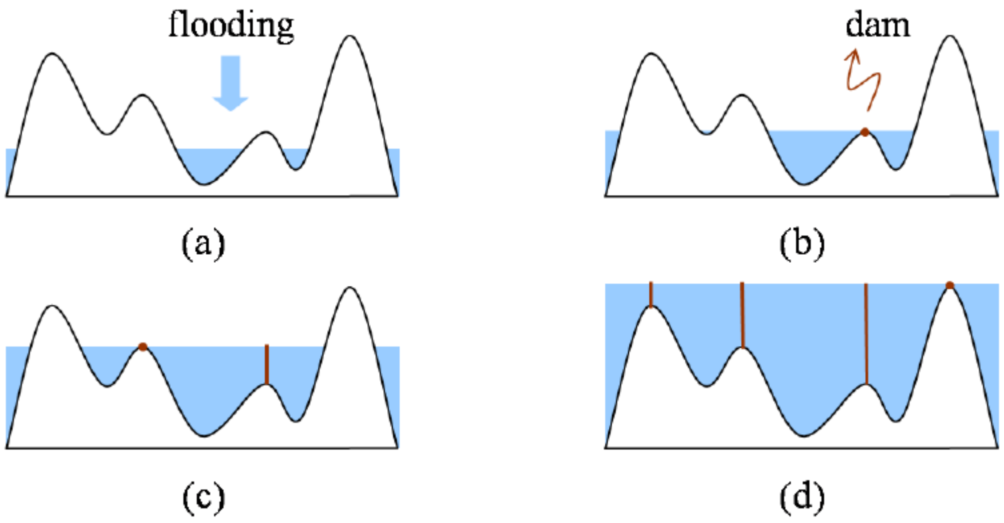


Figure 1: Watershed Algorithm

The algorithm starts to pour water in the topographic chart from the lowest basin to the highest peak. In the process, we may detect some peaks disjoined the catchment basins, called as dam.This concept is depicted in Fig.1. In Fig.1(a), water levels starts rising. Fig.1(b) and (c) shows dam locations (where two catchment areas are getting merged). Final stage is shown in Fig.1(d) which shows three “dam”s.

Outline of Watershed segmentation algorithm is as follows:

- Punch the regional minimum and flood the entire topography at uniform rate from below.
- A dam is built to prevent the rising water from distinct catchment basins from merging.
- Eventually only the tops of the dams are visible above the water line.
- These dam boundaries correspond to the divide lines of the watersheds.

Catchment basins will represent segments of image while dam boundaries will be edges in it. Though, this algorithm will give segments with continuous boundaries, it will also give rise to over-segmentation.

1.1.3 Minimum spanning tree based segmentation

Felzenszwalb et. al.[4] introduced a segmentation method based on Kruskal’s MST algorithm. Edges are considered in increasing order of weight; their endpoint pixels are merged into a region if this doesn’t cause a cycle in the graph, and if the pixels are ’similar’ to the existing regions’ pixels.

Felzenszwalb et. al. defined two measures: Internal Difference (Int) and Minimum Internal difference($MInt$). The internal difference of a component $C \subseteq V$ to be the largest weight in the minimum spanning tree of the component, $MST(C, E)$.

$$Int(C) = max_{e \in MST(C,E)} w(e) \tag{1}$$

It means, any component C remains connected only if edges with weight at least $Int(C)$ are kept. Minimum Internal difference($MInt$), is defined as,

$$MInt(C1, C2) = min(Int(C1) + \tau(C1), Int(C2) + \tau(C2)) \tag{2}$$

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them. Felzenszwalb et. al. used a threshold function based on the size of the component.

$$\tau(C) = k/|C| \tag{3}$$

where $|C|$ denotes the size of C, and k is some constant parameter. So, if component size is small, it will require larger evidence for a boundary.

Here, we start with each vertex being a component itself. Now, we merge two components only if Minimum Internal Difference between those two components is at least as great as smallest weight of edge between those two components.
Outline of algorithm is as follows:

Input :- Graph $G(V, E) : |V| = n, |E| = m$
Output :- Set of Components $S = \{C_1, C_2, \dots C_r\}$

```
1. Sort set of edges, E into $\pi = \{o_1, o_2, \dots o_m\}$, by non-decreasing edge weight.
2. Let initial segmentation $S^0 = \{\{v_i\}\} i = 1 \dots n$ i.e. each vertex is in its component.
3. Repeat step 4 for $q = 1, ..., m$.
4. Form next level segmentation $S^q$ from earlier segmentation $S^{q-1}$ as follows
    • Let $v_i$ and $v_j$ be end vertices of edge $o_q$ i.e. $o_q = (v_i, v_j)$
    • Let $C_i^{q-1}$ = the component in $S^{q-1}$ containing $v_i$
    • Similarly, $C_j^{q-1}$ = the component in $S^{q-1}$ containing $v_j$
    • If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then
          Merge $C_i^{q-1}, C_j^{q-1}$ in $S^q$.
      Otherwise $S^q = S^{q-1}$
5. Return $S = S^m$
```

In Grid graphs weights of edges can be given by difference of intensities between two pixels, while nearest neighbor graphs weights of edges are given by euclidean distance between two pixels represented in feature space such as (x, y, r, g, b).

Currently, Graph cuts based methods has emerged as a preferred way to solve image segmentation problem. We will discuss some popular methods based on Graph cuts in following sections.

2 Energy Minimization using Graph Cuts

Image Segmentation problem can be modeled as Energy minimization of MRF with Grid Graph containing image pixels. In this framework, one seeks the labeling f that minimizes the energy,

$$E(f) = E_{smooth}(f) + E_{data}(f) \tag{4}$$

Here $E_{smooth}(f)$ measures the extent to which f is not piecewise smooth, while $E_{data}(f)$ measures the disagreement between f and the observed data. Many different energy functions have been proposed in the literature. The form of $E_{data}(f)$ is typically,

$$E_{data}(f) = \sum_{p \in P} D_p(f_p) \tag{5}$$

where D_p measures how well label f_p fits pixel p. Mostly used model for E_{smooth} is Potts model given as,

$$E_{smooth}(f) = \sum_{\{p,q\} \in N} u_{\{p,q\}}.T(f_p \neq f_q) \tag{6}$$

where T is indicator function i.e. it will output 1 if the input condition is true.

So, in Potts Model, discontinuities between any pair of labels are penalized equally. This is, in some sense, the simplest discontinuity preserving model.

2.1 Approximation via Graph cuts

Energy-based methods attempt to model some global image properties that cannot be captured, for example, by local correlation techniques. The main problem, however, is that interesting energies are often difficult to minimize. e.g. It is proved that minimization of Potts energy on Grid Graph with multiple labels is NP hard.

So, to find approximate solution, Boykov et.al[3] proposed two algorithms named, $\alpha - \beta$ swap and α-expansion. In contrast to the standard moves(single pixel changing its label at a time), these moves allow a large number of pixels to change their labels simultaneously. This makes the set of labellings within a single move of a locally optimal f exponentially large. Also, using these moves solutions do not change significantly by varying the initial labellings.

2.2 New Moves

Any labeling f can be uniquely represented by a partition of image pixels $\mathbf{P} = \{P_l | l \in L\}$ where $P_l = \{p \in P | f_p = l\}$ is a subset of pixels assigned label l.
Now, new moves are defined as follows:

- α-β swap
  Given a pair of labels α, β a move from a partition $\mathbf{P}$(labeling f) to a new partition $\mathbf{P}'$(labeling f') is called a $\alpha - \beta$ swap if $P_l = P'_l$ for any label $l \neq \alpha, \beta$. This means that the only difference between $\mathbf{P}$ and $\mathbf{P}'$ is that some pixels that were labeled α in $\mathbf{P}$ are now labeled as β in $\mathbf{P}'$, and vice-e-versa.

- α-expansion
  Given a label α, a move from a partition$\mathbf{P}$(labeling f) to a new partition $\mathbf{P}'$(labeling f') is called an α-expansion if $P_\alpha \subset P'_\alpha$ and $P'_l \subset P_l$ for any label $l \neq \alpha$. In other words, an α-expansion move allows any set of image pixels to change their labels to α but restrics pixels to retain their earlier labels whose original label is α.

2.3 α-β Swap Algorithm

α-β Swap Algorithm is shown in Fig. 2.

```
1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each pair of labels $\{\alpha, \beta\} \subset \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min E(f')$ among $f'$ within one $\alpha$-$\beta$ swap of $f$
   3.2. If $E(\hat{f}) < E(f)$, set $f$ := $\hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$
```

Figure 2: Swap Algorithm

We start with initial labeling either taken from seeds provided by user or computed from some algorithm. Now, for each pair of labels we find a labeling which is one $\alpha - \beta$ swap away from current labeling in a such a way that it will have minimum energy among all labellings which are one $\alpha - \beta$ swap away from current labeling. In this step, some pixels which are earlier labeled α may change their labels to β(and vice-versa), but pixels whose labels are neither α or β will continue to hold its earlier label. If this value(of minimum energy) is less than earlier minimum(computed for earlier pairs of labels) we set *success* flag to 1. This flag indicates that during iteration over pairs of labels we have found another minimum labeling over earlier minimum labeling, so we need to iterate again over all pairs of labels.

2.3.1 Optimal Swap Move

Given an input labeling f (partition P) and a pair of labels α, β optimal swap move will find a labeling $f^{'}$ that minimizes E over all labellings within one $\alpha - \beta$ swap of f. This is critical step in Swap Algorithm.

The structure of graph $G_{\alpha\beta}$ i.e. the subgraph of original image graph, with pixels which are labeled either α or β is shown in Fig. 3.

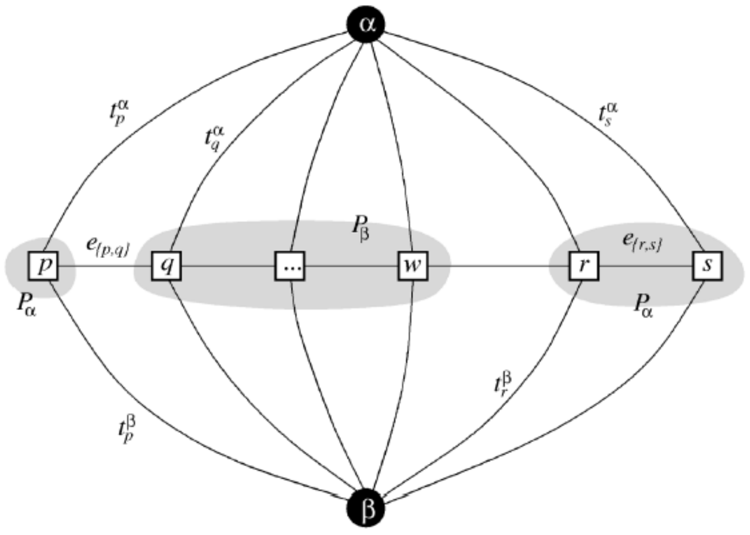


Figure 3: Structure of graph for swap move

For any image, the structure of $G_{\alpha\beta}$ will be as follows:

- The set of vertices includes the two terminals α and β, as well as image pixels p in the sets P_α and P_β. i.e. $V_{\alpha\beta} = \{\alpha, \beta\} \cup P_\alpha \cup P_\beta$
- Each pixel $p \in P_{\alpha\beta}$ is connected to vertices α and β by t-links t_p^α and t_p^β respectively.
- Each pair of pixels p, q which are neighbors($\{p, q\} \in \mathcal{N}$) are connected by n-links $e_{\{p,q\}}$.
- Weights of edges are as follows:

edge	**weight**	**for**
t_p^α	$D_p(\alpha) + \sum_{q \in N_p, q \notin P_{\alpha\beta}} V(\alpha, f_q)$	$p \in P_{\alpha\beta}$
t_p^β	$D_p(\beta) + \sum_{q \in N_p, q \notin P_{\alpha\beta}} V(\alpha, f_q)$	$p \in P_{\alpha\beta}$
$e_{\{p,q\}}$	$V(\alpha, \beta)$	$\{p, q\} \in \mathcal{N}, p, q \in P_{\alpha\beta}$

i.e. we weight t-links with Data term and n-links by Coherence term.

Now, for each pixel p any cut $\mathcal{C}$ on $\mathcal{G}_{\alpha\beta}$ should break exactly one t-link because if neither t-links are broken, there will be path between terminals and if we break both t-links, pixel p will be orphan. It gives natural labeling f^C corresponding to cut $\mathcal{C}$ as follows:

$$f_p^C = \begin{cases} \alpha & \text{if } t_p^\alpha \in \mathcal{C} \text{ for } p \in P_{\alpha\beta} \\ \beta & \text{if } t_p^\beta \in \mathcal{C} \text{ for } p \in P_{\alpha\beta} \\ f_p & p \notin P_{\alpha\beta} \end{cases} \tag{7}$$

i.e. we assign label α to p if cut $\mathcal{C}$ separates p from α, similarly p is assigned label β if cut $\mathcal{C}$ separates p from β. But we don't change labels of pixels whose labels are neither α nor β.

For pair of neighborhood pixels, p, q, cut $\mathcal{C}$ also sever $e_{\{p,q\}}$ only if it(cut) leaves those pixels connected to different terminals. It is depicted in Fig.4. We can see in Fig.4(a) and (b) pixels p

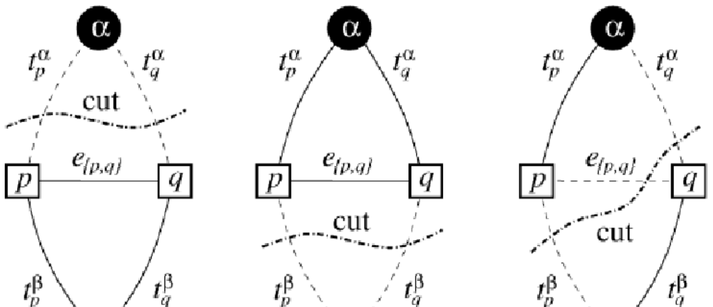


Figure 4: Properties for Swap move

and q are connected to same terminal, so there is no need to break n-link between those pixels. But, in Fig.4(c), if we don't sever n-link between p and q, there will be path between terminals.

In [3], it is shown that there is one to one correspondence between cuts $\mathcal{C}$ on $G_{\alpha\beta}$ and labellings that are one $\alpha - \beta$ swap away from f and the cost of cut is, $E(f^C)$.

2.4 α-Expansion Algorithm

α Expansion Algorithm is shown in Fig. 5. Expansion algorithm is very much similar to Swap algorithm with one difference that in each iteration(for each label) we try to find labeling which is one α-expansion from current labeling. In step 3.2, some pixels whose label is other than α(the label on which we are iterating over) may change their labels to α, but pixels whose label is α do not change their label. So, α-Expansion can be thought of special case of α-β Swap, where β is collection of all labels other than α and movement from only β to α is allowed.

```
1.  Start with an arbitrary labeling $f$
2.  Set success := 0
3.  For each label $\alpha \in \mathcal{L}$
    3.1.  Find $\hat{f} = \arg\min E(f')$ among $f'$ within one $\alpha$-expansion of $f$
    3.2.  If $E(\hat{f}) < E(f)$, set $f$ := $\hat{f}$ and success := 1
4.  If success = 1 goto 2
5.  Return $f$
```

Figure 5: Expansion Algorithm

2.4.1 Optimal Expansion Move

Given an input labeling f (partition P) and a label α, we would like to find a labeling $\hat{f}$ that minimizes E over all labellings within one α-expansion of f. The structure of graph for expansion move is shown in Fig. 6

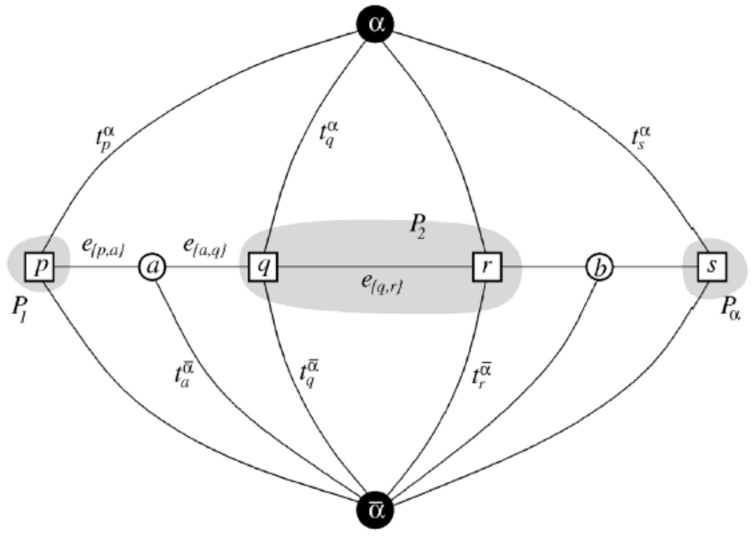


Figure 6: Structure of graph for expansion move

For any image, the structure of G_α for expansion move will be as follows:

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- In addition, for each pair of neighboring pixels $\{p, q\} \in N$ separated in the current partition (i.e.,such that $f_p \neq f_q$), we create an auxiliary node $a_{\{p,q\}}$.

$$V_\alpha = \{\alpha, \bar{\alpha}, \mathcal{P}, \cup_{\{p,q\} \in \mathcal{N}, f_p \neq f_q} a_{\{p,q\}}\}$$

- Each pixel $p \in \mathcal{P}$ is connected to the terminals α and $\bar{\alpha}$ by t-links t_p^{α} and $t_p^{\bar{\alpha}}$, respectively.
- Each pair of neighboring pixels p, q ($\{p, q\} \in \mathcal{N}$) such that $f_p = f_q$ are connected by n-links $e_{\{p,q\}}$.
- Each pair of neighboring pixels p, q ($\{p, q\} \in \mathcal{N}$) such that $f_p \neq f_q$ we create three extra edges, $\epsilon_{\{p,q\}} = \{e_{\{p,a\}}, e_{\{a,q\}}, t_a^{\bar{\alpha}}\}$. $e_{\{p,a\}}$ and $e_{\{a,q\}}$ are edges between auxillary node a and p, q respectively, while $t_a^{\bar{\alpha}}$ is edge between auxillary node and terminal $\bar{\alpha}$.
- Weights of edges are as follows:

edge	**weight**	**for**
$t_p^{\bar{\alpha}}$	∞	$p \in P_\alpha$
$t_p^{\bar{\alpha}}$	$D_p(f_p)$	$p \notin P_\alpha$
t_p^{α}	$D_p(\alpha)$	$p \in P$
$e_{\{p,a\}}$	$V(f_p, \alpha)$	$\{p, q\} \in \mathcal{N}, f_p \neq f_q$
$e_{\{a,q\}}$	$V(\alpha, f_q)$	
$t_a^{\bar{\alpha}}$	$V(f_p, f_q)$	
$e_{\{p,q\}}$	$V(f_p, \alpha)$	$\{p, q\} \in \mathcal{N}, f_p = f_q$

  i.e. we weight t-links with Data term except one exception that if a pixel is in α-partition we will set $t_p^{\bar{\alpha}} = \infty$ so that pixel will not go to other partition in any case. Edges connecting auxillary node are weighted in a such way that it will capture interaction among $f_p, f_q and \alpha$. Other n-links are weighted as coherence term between f_p and α.

Similar to Swap move, $\forall p \in P$, cut $\mathcal{C}$ should sever exactly one t-link, which gives natural labeling f^C as follows.

$$f_p^C = \begin{cases} \alpha & \text{if } t_p^{\alpha} \in \mathcal{C} \\ f_p & \text{if } t_p^{\bar{\alpha}} \in \mathcal{C} \end{cases} \quad \forall p \in P \tag{8}$$

i.e. we assign label α to p if cut $\mathcal{C}$ separates p from α, But we keep labels of pixels as in original if cut $\mathcal{C}$ separates p from $\bar{\alpha}$.

For pair of neighborhood pixels, p, q, if their labels are same($f_p = f_q$), cut $\mathcal{C}$ also sever $e_{\{p,q\}}$ only if it(cut) leaves those pixels connected to different terminals. But, for pair of neighborhood pixels, p, q, whose labels are different, we have to consider edges $\epsilon_{\{p,q\}} = e_{\{p,a\}}, e_{\{a,q\}}, t_a^{\bar{\alpha}}$. Depending upon which t-links are cut, zero or one of these edges needs to be severed.It is depicted in Fig.7.

If $t_p^{\alpha}, t_q^{\alpha} \in \mathcal{C}$, then we need not cut any of the edges from $\epsilon_{\{p,q\}}$ as there is no path between terminals as shown in Fig.7(a). Fig.7(b) shows that if $t_p^{\bar{\alpha}}, t_q^{\bar{\alpha}} \in \mathcal{C}$, we need to cut $t_a^{\bar{\alpha}}$ so that there is no path between terminals . We could have deleted $e_{\{p,a\}}$ and $e_{\{a,q\}}$ instead of just $t_q^{\bar{\alpha}}$, but it won't give us minimum cut due to triangle inequality($V(f_p, f_q) \leq V(f_p, \alpha) + V(\alpha, f_q)$). Finally, if $t_p^{\bar{\alpha}}, t_q^{\alpha} \in \mathcal{C}$, we need to cut $e_{\{p,a\}}$ to separate terminals as shown in Fig.7(c). Similarly, if $t_p^{\alpha}, t_q^{\bar{\alpha}} \in \mathcal{C}$, we need to cut $e_{\{a,q\}}$.

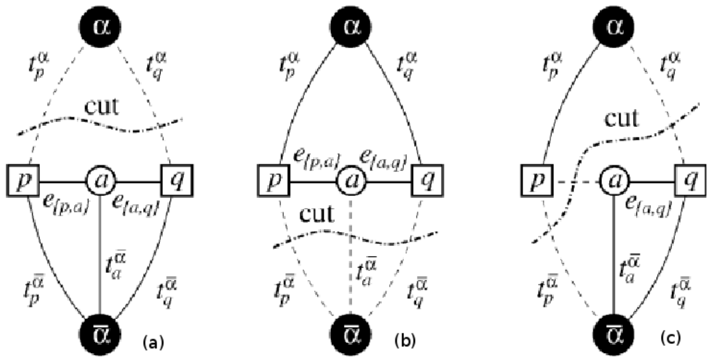


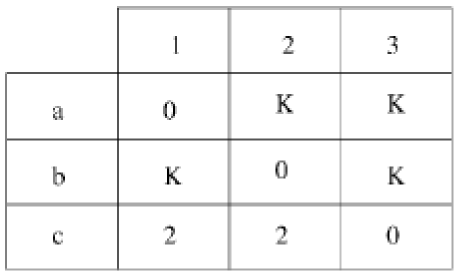
Figure 7: Properties for expansion move

Similar to Swap move, in [3] it is proved that there is one to one correspondence between elementary cuts on G_α and labelings within one α-expansion of f and cost of elementary cut $\mathcal{C}$ is $E(f^C)$.

2.5 Example

In this subsection, we will see a simple example which compares Swap move and Expansion move.

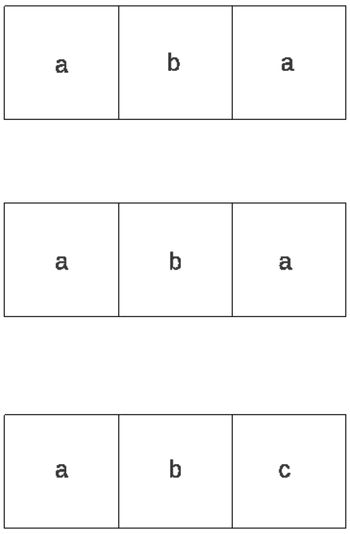
Assume, an image consisting of just three pixels 1,2 and 3 arranged in one row. There are possible three labels a, b, c. The values for D_p for each pixel/label combination are shown in Fig.8. There are only two neighboring pixel pairs, i.e.$\{1, 2\}$ and $\{2, 3\}$. Let $V(a, b) = V(b, c) = K/2$ and $V(a, c) = k$. Initially label all pixels as a.



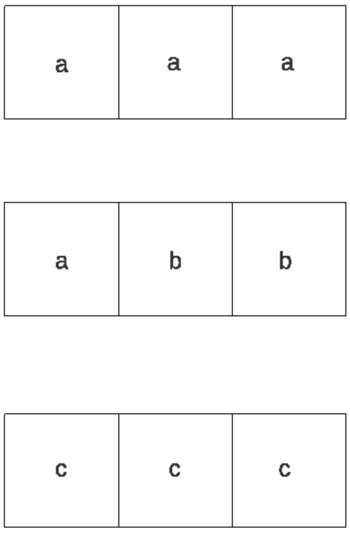
	1	2	3
a	0	K	K
b	K	0	K
c	2	2	0

Figure 8: D_p values

- Swap Moves:- We will process pairs of labels in order (a,b), (b,c) and then (a,c) for swap moves. Labeling corresponding to min-cut after each pair of labels are processed is shown in Fig 9a. Minimum energy for last configuration is K.
- Expansion Moves :- We will process labels in order a, b, c for expansion moves. Labeling corresponding to min-cut after each label is processed is shown in Fig 9b. Minimum energy for last configuration is 4.



(a) Labeling after Swap moves

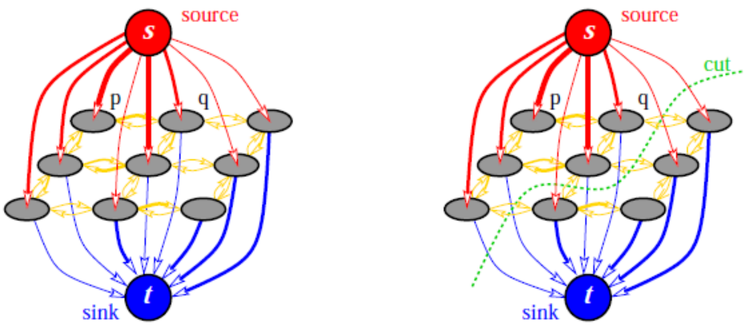


(b) Labeling after Expansion moves

3 Methods for Finding Min-cut in Flow Graphs

When we are solving energy minimization of grid-graphs using graph cuts, basic step is finding min-cut itself. In Figure 9(a), we show a simple example of a two terminal graph that can be used to minimize the Potts case of energy on a 3×3 image with two labels. An s/t cut C on a graph with two terminals is a partitioning of the nodes in the graph into two disjoint subsets S and T such that the source s is in S and the sink t is in T. Example of such cut is shown in 9(b).

There are mainly two algorithms for solving Max-flow/Min-Cut problem: Augmenting paths and Push-relabel. We will see special variations of these algorithms in following subsections.



(a) A graph $\mathcal{G}$ (b) A cut on $\mathcal{G}$

Figure 9: Example of directed capacitated graph

3.1 Boykov-Kolmogorov Algorithm

Normally, augmenting path based methods start a new breadth-first search for $s \to t$ paths as soon as all paths of a given length are exhausted. In the context of graphs in computer vision, building a breadth-first search tree typically involves scanning the majority of image pixels.

In [2], for finding augmenting path, two non-overlapping search trees S and T with roots at the source s and the sink t, correspondingly. Fig.10. depicts structure of these trees and corresponding augmenting path. To find different augmenting path, these trees are not rebuilt from scratch and they reuse these trees.

In tree S all edges from each parent node to its children are non-saturated, while in tree T edges from children to their parents are non-saturated. The nodes that are not in S or T are called free. The nodes in the search trees S and T can be either active or passive. The active nodes represent the outer border in each tree while the passive nodes are internal. Now, active nodes allow trees to grow by acquiring new children (along non-saturated edges) from a set of free nodes. An augmenting path is found as soon as an active node in one of the trees detects a neighboring node that belongs to the other tree.

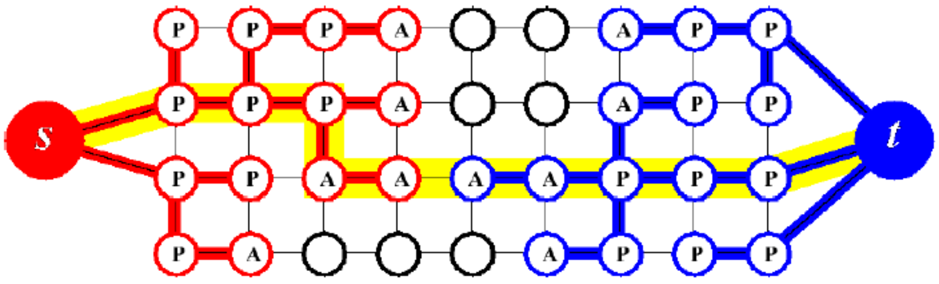


Figure 10: Example of search trees S(red nodes) and T(blue nodes)

This algorithm has following three stages:

- growth stage: search trees S and T grow until they touch giving an $s \to t$ path,
- augmentation stage: the found path is augmented, search tree(s) break into forest(s),
- adoption stage: trees S and T are restored.

3.1.1 "growth" stage

In growth stage, active nodes explore adjacent non-saturated edges and acquire new children from set of free nodes. Now, newly acquired nodes will become active members of corresponding search trees. When all neighbors of an active node are explored, it becomes passive. The growth stage terminates if an active node encounters a neighboring node that belongs to the opposite tree. In this case we detect a path from the source to the sink, as shown in Figure 10.

3.1.2 "augmentation" stage

The augmentation stage augments path found in growth stage. As we are pushing maximum possible flow through augmenting path, some edge(s) will become saturated. So, when we remove such edge(s), some of the nodes in the trees S and T may become orphans as the edges linking them to their parents are no longer valid. It effects split of the search trees S and T into forests. The source s and the sink t are still roots of two of the trees while orphans form roots of all other trees.

3.1.3 "adoption" stage

This stage is most important aspect of the whole algorithm. The objective of this stage is to restore single-tree structure of sets S and T with roots in the source and the sink. In this stage, we try to find valid parent for orphans created in **"augmentation"** stage. A new parent should belong to the same set, S or T, as the orphan. A parent should also be connected through a non-saturated edge. If there is no qualifying parent we remove the orphan from S or T and make it a free node.

After the adoption stage is completed the algorithm returns to the growth stage. The algorithm terminates when the search trees S and T can not grow (no active nodes) and the trees are separated by saturated edges. This implies that a maximum flow is achieved. The corresponding minimum cut can be determined by $\mathcal{S} = S$ and $\mathcal{T} = T$.

Though, theoretical worst case complexity of this algorithm is very high $\mathcal{O}(mn^2|C|)$(where m is number of nodes, n is number of edges and $|C|$ is cost of minimum cut), experimental comparison shows that this algorithm outperforms standard algorithms.

3.2 Voronoi based Preflow Push Algorithm

Push-relabel algorithm is one of the efficient ways to find Max-flow/Min-cut in Flow graphs. In this algorithm, we "push" maximum possible flow from source towards sink through intermediate nodes whose "height" is less than source. When there is no possibility of "push", we "relabel" nodes with new heights. Algorithm converges when no more push or relabel operations are possible.

C. Arora et al.[1], developed a new algorithm based on push-relabel method, called as **Voronoi based Preflow Push**(VPP). The structure of the graph used is similar to Fig.9. Flows in t-links are set equal to their capacities, and in n-links those are set to zero. Now if we label the vertices other than s and t by their excesses($inflow - outflow$), and delete all t-links, it is easy to see that the original max-flow problem is equivalent to finding Max-flow in resultant graph. Source-Sink max-flow problem is solved in the resultant flow graph by treating nodes with positive excesses as sources and those with negative excesses as sinks.

The main steps of this algorithm are as follows:

1. Initialization
2. Create Voronoi region graphs around sink clusters.
3. In each Voronoi region, first push flow from source cluster to sink cluster boundary and then push flow within sink cluster.
4. Rebuild Voronoi region graphs around remaining sink clusters.

3.2.1 Initialization

For all source and sink nodes $excess(v)$ is set equal to source capacity if v is a source or equal to negative of sink capacity if v is a sink. With every node v of the grid graph we associate label d(v) called distance label. Distance labels are shortest distance to a sink node on a sink cluster boundary. Initially these labels are generated by the standard global labeling procedure of push relabel algorithms. Initialization also inserts all sources in structures called *Excess List (EL).*

Excess lists are maintained for each distance label. $EL(d)$ contains source nodes with distance label d. d_{max} is the largest distance label assigned to any node.

3.2.2 Creation of Voronoi Region Graphs

Collection of nodes in which there is a path between any two nodes passing through only nodes in the collection is called a *cluster*. In segmentation problem, sources and sinks forms clusters, and such clusters are often interspersed. So, we will create Voronoi region graphs around each sink cluster as shown in Fig.11. Each Voronoi region contains one sink cluster and one or more source clusters which are reachable from given sink cluster. In Fig.11, sink clusters are shown as red ovals while source clusters are shown as blue ovals. "Push flow" operation is performed per Voronoi Region.

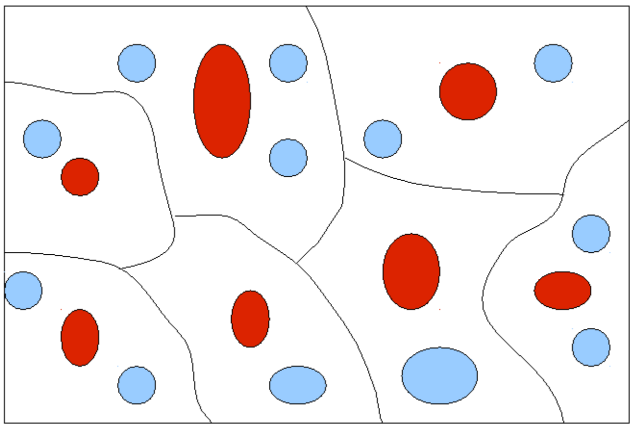


Figure 11: Example of Voronoi Region Graph

3.2.3 Push Flow

Push flow happens in two phases. In first phase, excess from sources is pushed to the boundaries of sink cluster in topological order. i.e. first we will process nodes from $EL(d_{max})$, then $EL(d_{max}-1)$ and so on. From a node, flow is pushed saturating the out edges till the node has no excess left or all out edges of the node get saturated. The saturated edges are deleted and a node whose all out-edges are deleted is inserted in a list called *Disconnected List(DL)*.

Second phase moves the excess(which is accumulated at boundary nodes of a sink cluster), inside the sink cluster. Second phase starts from those sink cluster boundary nodes with positive excess on them and pushes the excess inside the cluster in a breadth first manner.

3.2.4 Rebuilding Voronoi Regions

A node v is put in disconnected list during the Push flow stage because all paths from v to the sink of a Voronoi region graph have been saturated and there is no remaining path from v to a sink in the Voronoi region graph on which flow can be pushed. Similarly, nodes in the Voronoi region graphs for which all paths to a sink pass through nodes which are already put in DL are also effectively disconnected.

In **Rebuild** operation, first we identify all such nodes whose distance labels need to be corrected. An augmenting path from a node v in $DL(d)$ to a sink in the new residual graph will necessarily have to pass through a node which has its path to sink intact i.e. it has retained its shortest distance label after a push flow stage.

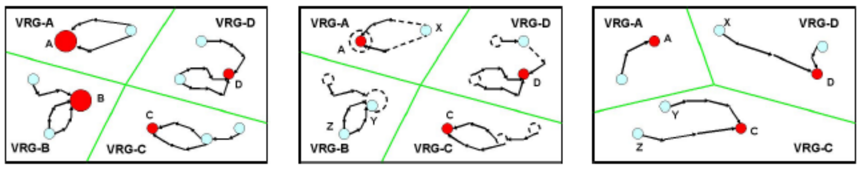
If such path exists from node v(whose all out edges are saturated in last push phase) will have to pass through a neighbor w not in its out-edge list as it existed in last push phase. For such node w, its label $d(w)$ was greater than or equal to $d(v)$ in the Voronoi region graph in which flow was pushed (otherwise flow would have pushed to this node in last push phase). So, now we have to increase label of v to $d(w) + 1$. This is achieved as follows:

1. Per distance label, create Rebuild list(RL) containing nodes which will be new neighbors of disconnected nodes.

2. Iterate over each rebuild list starting from lowest distance label Rebuild List, $RL(d)$

```
• For each $w$ in $RL(d)$,
    If there is residual edge$(v, w)$ and $residual(v, w) > 0$ and $v$ is disconnected
        Set $d(v) \leftarrow d + 1$
        make $(v, w)$ out-edge of $w$ and in-edge of $v$
        Put $v$ in $RL(d + 1)$
```

It is possible that in the process of rebuilding, a node shifts from one Voronoi region to another. This will depend upon the Voronoi region to which the node in the rebuild list through which the disconnected node finds a new path to a sink belonged.



(a) After initialization/rebuild (b) After Push flow (c) After rebuilding Voronoi regions

Figure 12: Flow Graph states

Figures 12(a),12(b), and 12(c) depict the state of the flow graph prior to a push flow iteration, after the push flow iteration, and after the corresponding rebuild phase respectively. In these figures sinks clusters are circles labeled A,B,C, and D. Rest of the circles are source clusters. Fig.12(a) depicts scenario prior to push phase with Four Voronoi regions corresponding to sink clusters A,B,C, and D.

Fig. 12(b) represents the state after the push flow phase. Thick dashed lines indicate saturated edges while dashed circles show the changes in sources and sink clusters. We can see that in VRG-A, sink cluster is shrunk, while in VRG-B and VRG-D sink clusters are not changed but few source clusters are disappeared and in VRG-C sink cluster is disappeared and new source cluster is created.

Fig.12(c) represents the state of the flow graph immediately after the rebuild phase. The three Voronoi regions correspond to the remaining sink clusters. Voronoi boundaries have shifted and sources X, Y, and Z are now in different Voronoi regions.

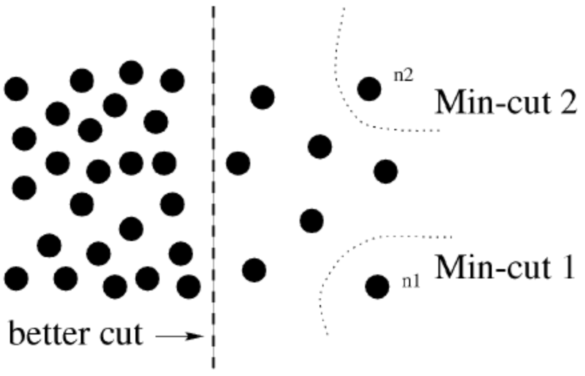


Figure 13: Example showing "bad" minimum cuts

4 Normalized Graph Cuts

For a graph $G(V, E)$ cost of cut which partitions graph into two parts A and B is defined as,

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{9}$$

where $w(u, v)$ is weight of edge (u, v)
The minimum cut criteria favors cutting small sets of isolated nodes in the graph.

Fig.13 illustrates one such case. Assuming the edge weights are inversely proportional to the distance between the two nodes, we see the cut that partitions out node n1 or n2 will have a very small value. In fact, any cut that partitions out individual nodes on the right half will have smaller cut value than the cut that partitions the nodes into the left and right halves.

4.1 Normalized Cut

To avoid this unnatural bias for partitioning out small sets of points, [7] proposes a new measure of disassociation between two groups. Instead of looking at the value of total edge weight connecting the two partitions, this metric computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. This disassociation measure is called as Normalized cut(Ncut). Formally,

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{10}$$

where $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$ measures overall association of partition A with whole graph. Now, the cut that partitions out small isolated points will no longer have small Ncut value, since the cut value will almost certainly be a large percentage of the total connection from that small set to all other node.

Similarly, we can define a measure for total normalized association within groups for a given partition

$$Nassoc(A, B) = \frac{assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, B)}{assoc(B, V)} \tag{11}$$

where $assoc(A, A)$ and $assoc(B, B)$ are total weights of edges connecting nodes within A and B, respectively. From (10) and (11) we can find relation between Ncut and Nassoc as,

$$Ncut(A, B) = 2 - Nassoc(A, B) \tag{12}$$

4.1.1 Finding Optimal Partition

Let $\mathbf{x}$ be $N = |V|$ dimensional vector containing elements x_i, $x_i = 1$ if i^{th} node $\in A$ and -1 otherwise.
Let $d(i) = \sum_j w(i,j)$ be sum of weights of all edges from node i to all other nodes.
Let $\mathbf{D} = N \times N$ diagonal matrix with $\mathbf{d}$ on its diagonal
Let $\mathbf{W} = N \times N$ symmetric matrix, $W(i,j) = w_{ij}$
Let $\mathbf{1}$ be an $N \times 1$ vector of all ones.

Now, min $Ncut$ is given by,

$$min_{\mathbf{x}} Ncut(\mathbf{x}) = min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \tag{13}$$

with constraint that $y^T\mathbf{D1} = 0$ and $y(i) \in \{1, -b\}$.
where $\mathbf{y} = (1+\mathbf{x}) - b(1-\mathbf{x}), b = \frac{k}{1-k}, k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i}$.
If $\mathbf{y}$ is relaxed to take on real values, we can minimize (13) by solving the generalized eigenvalue system,

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y} \tag{14}$$

It is proved that the second smallest eigenvector of the generalized eigensystem (14) is the real valued solution to normalized cut problem. Similarly we can show that the eigenvector with the third smallest eigenvalue is the real valued solution that optimally subpartitions the first two parts. We can extended this argument to show that one can subdivide the existing graphs, each time using the eigenvector with the next smallest eigenvalue.

4.2 Grouping Algorithm

Grouping algorithm consists of following steps:

1. Construct a weighted graph $G = (V, E)$ by taking each pixel as node and connecting each pair of pixels by edge. The weight of edge should reflect similarity between two pixels. e.g. In case of brightness image, weight of edge connecting two nodes i and j can be defined as:

$$w_{ij} = e^{\frac{-\|F_{(i)}-F_{(j)}\|^2}{\sigma_I^2}} * \begin{cases} e^{\frac{-\|X_{(i)}-X_{(j)}\|^2}{\sigma_X^2}} & \text{if } \|X\,(i) - X\,(j)\| < r \\ 0 & \text{otherwise} \end{cases}$$

   where $X(i)$ is the spatial location of node i, and $F(i)$ is a feature vector based on intensity, color, or texture information at that node.

2. Solve for the eigenvectors with the smallest eigenvalues of the system (14). Generally it has O(n^3) time complexity. But as graph is locally connected resulting eigensystems are very sparse. Also, we need only top few eigenvectors with very less precision. Hence, complexity of this operation can be reduced significantly using Lanczos method.

3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph. Ideally eigenvector should only take two discrete values and the signs of the values can tell us exactly how to partition the graph. However, in practice, eigenvectors can take on continuous values and we need to choose a splitting point to partition it into two parts. Possible

choices for split points are 0, mean or median value. Alternatively, we can search for the splitting point such that the resulting partition has the best $Ncut(A, B)$ value.

4. Decide if the current partition should be subdivided by checking the stability of the cut, and make sure $Ncut$ is below the prespecified value. Stability criterion measures the degree of smoothness in the eigenvector values. Higher the smoothness, lower the confidence of finding optimal partition. So we will not use such eigenvectors for partitioning.

5. Recursively repartition the segmented parts if necessary. Alternatively, one can use all of the top eigenvectors to simultaneously obtain a K-way partition.

Since the normalized association is exactly $2 - Ncut$, the normalized cut value, the normalized cut formulation seeks a balance between the goal of clustering and segmentation

5 Conclusion

Though there are many approaches for solving Image Segmentation problem, currently Graph based methods are preferred over other approaches. Boykov et. al.[3] posed Image Segmentation problem as Energy minimization problem in MRF with image Grid-graphs and found approximate solution using Graph cuts. In [3] new moves namely, $\alpha - \beta$ swap and α expansion were proposed. These moves allow large number of pixels to change their label as opposed to single pixel changing its label in standard moves. It is proved that using α expansion moves, we can approximately reach optimal configuration in very less time.

Main step in Flow based Graph cut method is to find Min-cut itself. In [2], new augmenting path based method is proposed. This method has given an efficient way to find augmenting path as we don't have to start from scratch while finding new augmenting path but use existing structure itself with minor arrangements. C. Arora et.al. [1] has proposed an algorithm namely "Voronoi based Push Preflow"(VPP) based on Push-relabel method. VPP algorithm exploits structural properties inherent in image grid-graphs. It pushes flow in phases while relabeling is done only in global manner.

In [7], new approach called as "Normalized cuts" is proposed which focuses on extracting the global impression of an image rather than focusing on local features and their consistencies in the image data. *Normalized cut* criterion measures both the total dissimilarity between the different groups as well as the total similarity within the groups. Is is showed that efficient computational technique based on a generalized eigenvalue problem can be used to optimize this criterion.

Acknowledgements

I thank Prof. Sharat Chandran for guiding me in the seminar and introducing me to this field. Our valued discussions on various topics related to seminar have enriched me a lot.

References

[1] Chetan Arora, Subhashis Banerjee, Prem Kalra, and S. Maheshwari. An efficient graph cut algorithm for computer vision problems. In *Computer Vision ECCV 2010*, volume 6313 of *Lecture Notes in Computer Science*, pages 552–565. Springer Berlin / Heidelberg, 2010.

[2] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:1124–1138, 2004.

[3] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1222–1238, 2001.

[4] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–182, 2004.

[5] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100 – 132, 1985.

[6] Nikhil R Pal and Sankar K Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9):1277 – 1294, 1993.

[7] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.