Introduction to Computer Vision: Segmentation by Clustering

Navasardyan Shant

October 27, 2019

Overview

Overview

What is Segmentation?

Segmentation

Any kind of grouping the pixels in an image we call **image segmentation**.

Grouping Criteria

For image segmentation there can be many grouping criteria, for example [a]

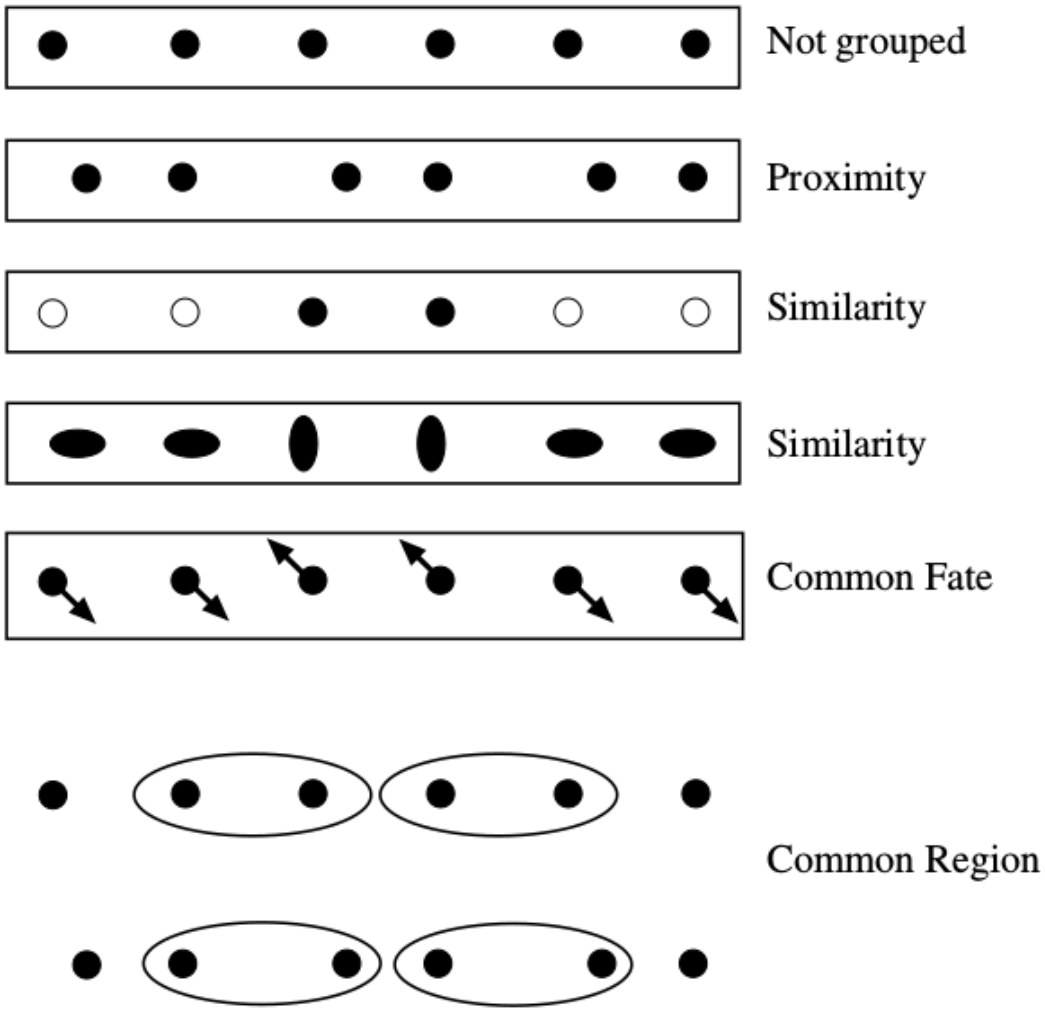
- **Proximity**: Tokens that are nearby tend to be grouped;
- **Similarity**: Similar tokens tend to be grouped;
- **Common Fate**: Tokens that have coherent motion tend to be grouped;
- **Common Region**: Tokens that lie inside the same closed region tend to be grouped.

Here by token we assume a group of pixels (particularly it can be a pixel).

[a] Forsyth, Ponce: 'Computer Vision: A Modern Approach', 2nd edition, 2012

What Is Segmentation?

Here are some examples of grouping tokens in an image:



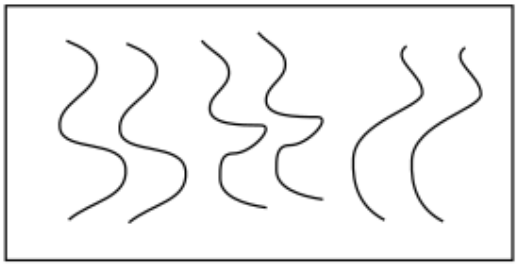
What is Segmentation?

Here are some other grouping criteria for parts of an image:

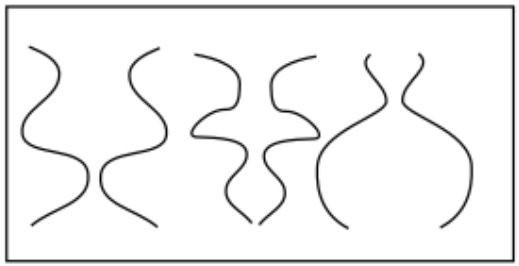
Other Grouping Criteria

- **Parallelism**: Parallel curves or tokens tend to be grouped;
- **Closure**: Tokens or curves that tend to lead to closed curves tend to be grouped;
- **Symmetry**: Curves that lead to symmetric groups are grouped together;
- **Continuity**: Tokens that lead to continuous (as in joining up nicely, rather than in the formal sense) curves tend to be grouped.

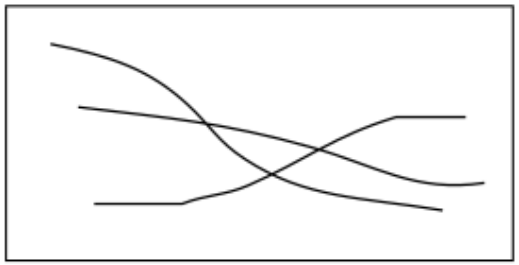
What Is Segmentation?



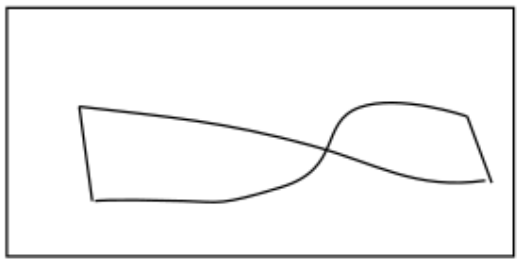
Parallelism



Symmetry



Continuity



Closure

Overview

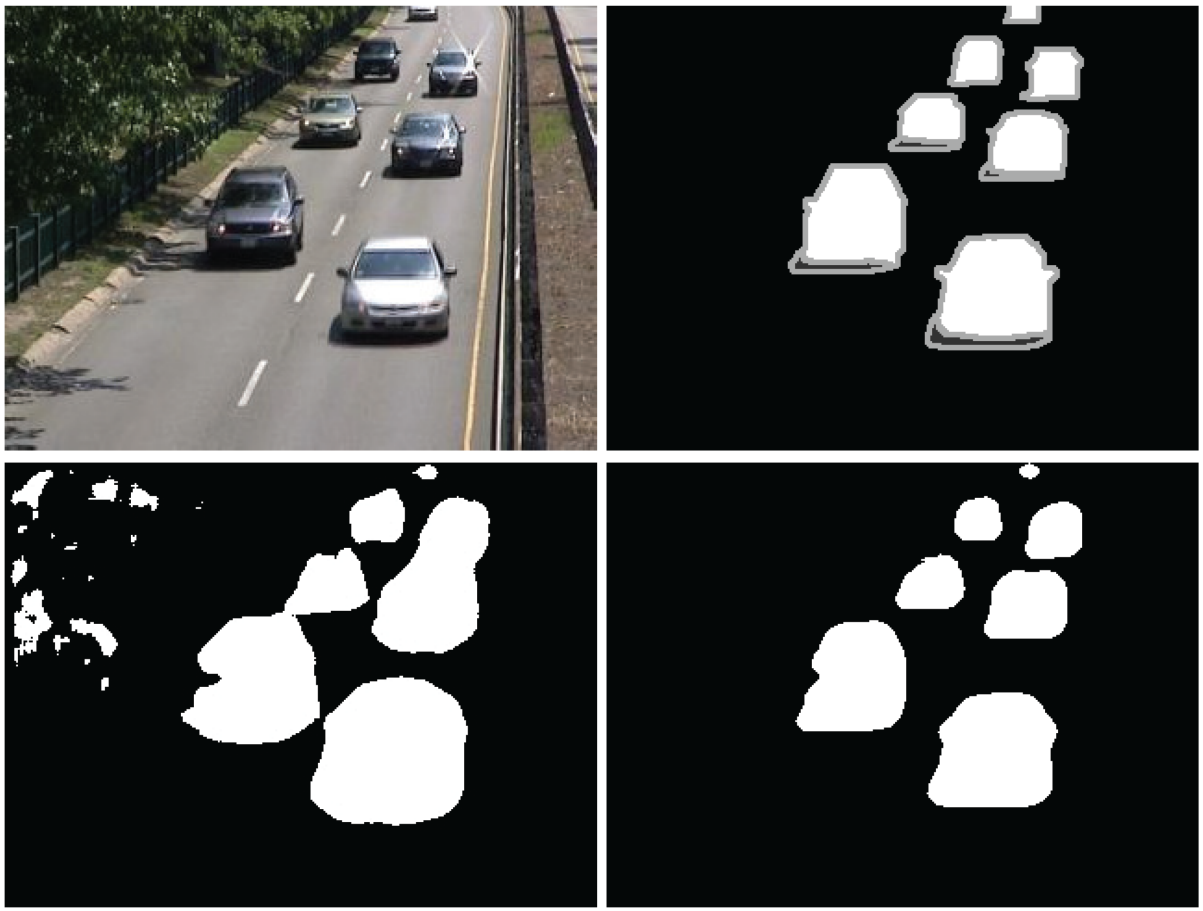
Applications

In this section we will talk about the following types of applications of image segmentation.

- Background Subtraction.
- Shot Boundary Detection.
- Interactive Segmentation.
- Segmentation to Form Image Regions.

Background Subtraction

In many applications we need to group the foreground pixels in the case of a stable background. Standard examples are detecting items on a conveyor belt or counting motor cars in an overhead view of a road.



Background Subtraction

Note

In some cases background is not exactly the same (for example during the day the background can be changed due to lightness). But in many cases the changing background is changing smoothly, so for each moment we can take the *moving average* of all past backgrounds.

Recall

In general the weighted moving average of $x_1, x_2, \ldots, x_N$ is calculated by $E_N = \frac{\omega_a x_N + \sum_{i=1}^{N-1} \omega_i x_{N-i}}{\omega_c}$ for some weights.
A commonly used moving average is **exponential moving average**:

$$E_0 = x_1, \qquad E_N = \alpha x_N + (1-\alpha) E_{N-1},$$

where α is some parameter from $[0, 1]$.

Background Subtraction

Algorithm

So the algorithm of background subtraction is the following.

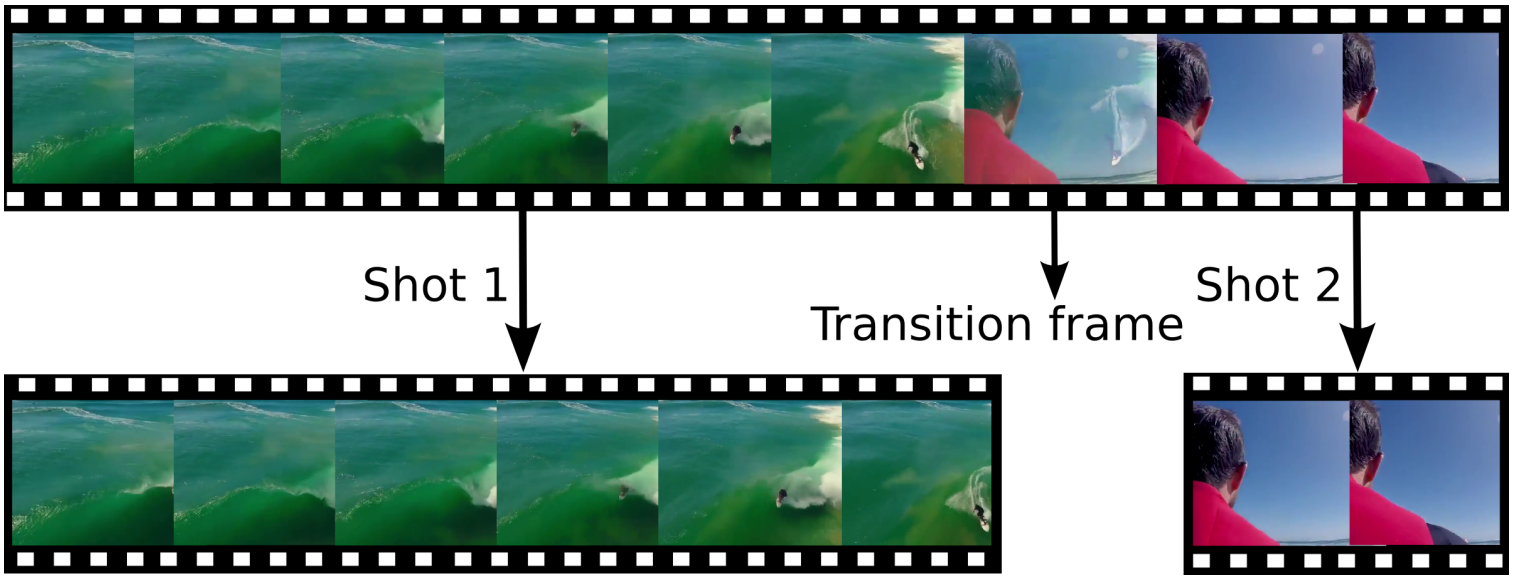
- Form an initial background B_0.
- For the frame x_t at the moment t compute the background estimate B_t with moving average (for example with exponential moving average with a small α).
- Compute $M = |x_t - B_t|$.
- Take a pixel of M as a foreground pixel if the value of this pixel in M is more than some predefined threshold.

Exercise

Implement the Background Subtraction algorithm.

Shot Boundary Detection

Another problem which can be solved by a similar way such the background subtraction problem is the problem of *Shot Boundary Detection*.



The problem is to detect the frames in a video, which are significantly differ from their previous frame in video sequence. It can be useful for example for representing a video by its "key-frames" - one frame for one scene.

Shot Boundary Detection

Though this is a task of detection, we can treat it as a task of segmentation by considering the video as one big image composed of frames.

Algorithm

For each frame x_t (starting from $t = 1$) in video sequence $\{x_t\}_{t=0}^{T}$, compute a distance between x_t and its previous frame: $d = dist(x_t, x_{t-1})$. If d is larger than some predefined threshold, then take the frame x_t as a boundary frame. The distance between two frames can be computed in different ways (e.g. MSE, histogram-based, edge-differencing, etc.)

Interactive Segmentation

So far we have considered applications connected with videos. But people very often want to cut some object from one image and put it on another image.
For cutting an object from an image one needs to determine the pixels of this object. (Semi-)Automatically determining this pixels is a segmentation problem on an image: to segment the image into two regions, namely into *foreground* and *background*.

Interactive Cases

In many applications a user allowed to make some interaction to the segmentation algorithm, i.e. predefine roughly some foreground and background pixels, or give an approximate coarse edges of the object, etc.

Here are some examples of such interactions.

Intelligent Scissors

In one of interaction-based applications, a user sketches a curve close to the boundary of an object, then the algorithm moves this curve to boundary of the object. This algorithm is called the **Intelligent Scissors** algorithm. After a closed curve is obtained by the algorithm, one can take the inner part of this curve as a result of segmentation.

Painting Interface

Very often when cutting an object (i.e. in case of classifying pixels into two classes, namely foreground and background), a user is able to provide some information about some pixels. Trough this information (e.g. this information can be given by brush paintings on the image) a user provides the following types of pixels:

- Sure Background: The user provides some pixels, that are for sure background pixels.
- Sure Foreground: The user provides some pixels, which are for sure foreground pixels.
- Unsure background: The user provides some pixels which are most likely background pixels, but we are not sure for this.
- Unsure foreground: The user provides som pixels which are most likely foreground pixels, but we are not sure for this.

Grabcut Interface

Sometimes only a part of this information is provided by the user. For example in the paper " "GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts" [1], in first step user provides a bounding box of the object which is needed to be cut (then some user edits can be done also to improve the result). This bounding box can be treated as unsure foreground region, and the other part - sure background region.

(a) Selection marked by the user

(b) Segmented image

[1] https://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf

α-Matting and Brush Strokes Interface

Sometimes, when we want to extract foreground from the image, some pixels are neither pure background nor pure foreground. For example in a selfie-image, some pixels in hair region are not pure hair pixels (thus can be treated as some "mixes" of some foreground and background pixels).

α-Matting and Brush Strokes Interface

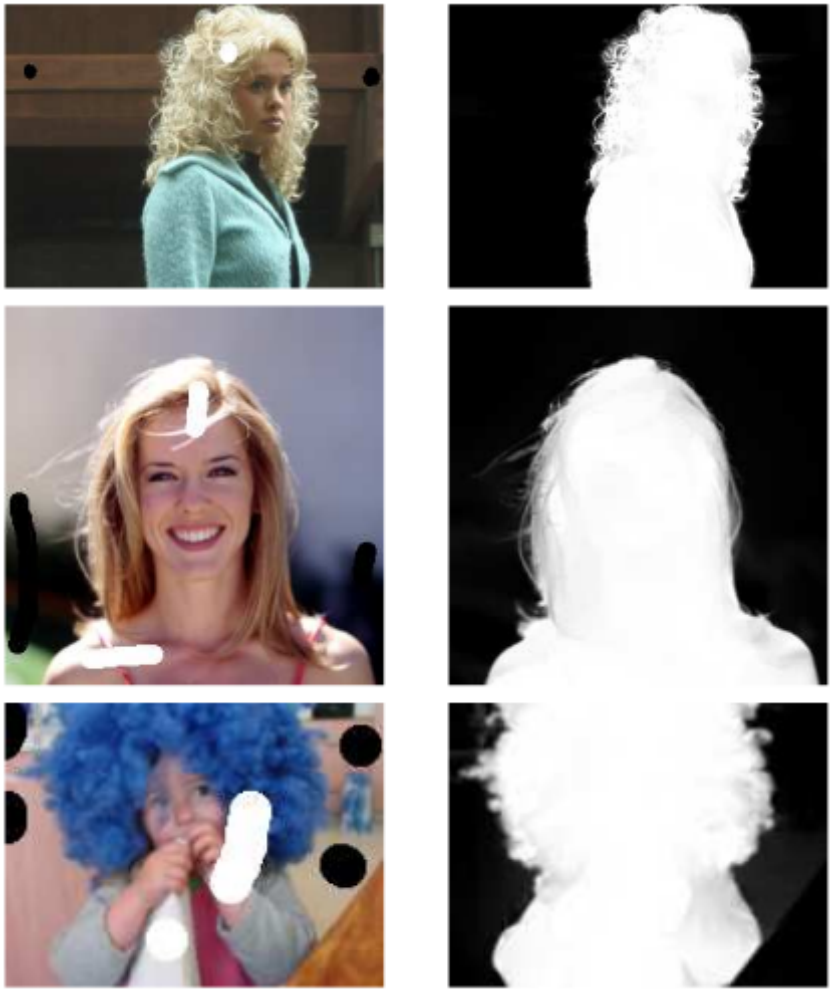
In this case for cutting an object we need to prepare a *matte*, an image (1-channeled image with values in range $[0, 1]$) traditionally denoted by α, for which α_i is the probability of being the i^{th} pixel a foreground pixel. Also, we can assume that the image I was obtained from some images $\mathbf{f}$ (foreground) and $\mathbf{b}$ (background) with the weights α (matte):

$$I = \alpha \cdot \mathbf{f} + (1 - \alpha) \cdot \mathbf{b}.$$

Figure: The alpha-matte of the image above

α-Matting and Brush Strokes Interface

Some algorithms use the user interactions to get the alpha-matte of an image. One of this algorithms, described in the paper "Sepctral Matting", uses sure foreground and sure background information, which is provided by the user as brush strokes:



Project website: http://www.vision.huji.ac.il/SpectralMatting/

α-Matting and Trimap Image Interface

Trimap

In case when the user provides only *sure foreground* and *sure background* pixels, we can treat the rest part as *unsure foreground (or background)*. Thus in this cases for the image we have a partition into three regions (clusters): sure foreground, sure background and unsure part. Images, which have values $0, 1$ and $\frac{1}{2}$ for this regions respectively, are called **Trimap Images** (of the initial image).

So in some cases of user-interaction-based segmentation algorithms, a trimap image is provided by the user

α-Matting and Trimap Image Interface

Some algorithms are using a trimap image provided by the user for getting the *alpha*-matte of an image ("A Closed Form Solution to Natural Image Matting"[2]).

Figure: original image, trimap, alpha-matte, composition of foreground and background with alpha-matte

[2] http://webee.technion.ac.il/people/anat.levin/papers/Matting-Levin-Lischinski-Weiss-CVPR06.pdf

Forming Image Regions

One application of segmentation is to partition an image into some regions, based on similarity of pixels (this similarity can be defined by different ways). This application is a quite big part of image segmentation, and often by saying segmentation one means partitioning an image into regions.

Forming Image Regions

In this course we will cover the following algorithms for image regions formation:

- K-means Clustering
- Agglomerative Clustering with Graphs
- Watershed Algorithm
- Normalized-Cut Algorithm

Overview

Clustering in Segmentation

Clustering

Clustering is a process of partitioning the data into some subsets in which elements are similar to each other (in some sense), and elements of different classes are dissimilar (in the same sense).

We also can treat an image as some data, which we need to cluster. In this case the task of segmentation (here in sense of partitioning the image) becomes to the task of clustering an image data.

Questions

So two main questions arise:

- How to define a relevant data for an image?
- How to choose an appropriate clustering algorithm?

Clustering

We answer this questions every time we describe a clustering method for image segmentation. Since in many cases the answer of the first question is a data of some feature-vectors for each pixel, the choice of the clustering algorithm is universal for the choice of this features. Hence we start with the answers to the second question.
Before starting to dive in clustering algorithms for image segmentation, we divide clustering algorithms into two abstract classes: **Agglomerative Clustering** and **Divisive Clustering**.

Agglomerative and Divisive Clustering

Agglomerative Clustering

In agglomerative clustering, in the first step, each data-point is treated as a cluster, and recursively, this clusters are merged:

- Make each point a cluster
- Until some **clustering criterion** is satisfied, merge the two **closest** clusters.

Divisive Clustering

In divisive clustering, in the first step the whole data is treated as one big cluster, and recursively is divided into small-ones:

- Make a cluster with all points
- Until some **clustering criterion** is satisfied, split a cluster into two subsets which have the largest **distance** among all splits.

Note that the *criterion* when to stop and *distance* between clusters are playing a huge role in this schemes of clustering algorithms.

Segmentation Using K-means

If we know that we need to divide our image into k segments, then we can use k-means clustering algorithm on feature-vectors associated with pixels.

Color as Feature

In case of segmenting a colored image (actually, this method is applicable in case of any number of channels), one can take all pixels as vectors in the space $\mathbb{R}^3$ and get clusters by k-means algorithm.

In this algorithm we do not force clusters to become connected, hence we can not expect their connectivity.

Color and Position as Feature

In order to make clusters to be more connected, we can work in the space $\mathbb{R}^5$, i.e. to each pixel we can assign its color (3 numbers) and its position (2 numbers), then apply k-means clustering algorithm.

Segmentation Using K-means

Exercise

Implement the k-means clustering algorithm for image segmentation, then assign to each pixel the color of the mean of that pixel's cluster. The obtained image is called **vector-quantized** image, and the process of this assignment is called **vector-quantization**.

Recall Some Definitions

Now we will discuss an agglomerative clustering algoritm for image segmentation.
For this let us recall some terminology from graph theory.

Definition (Graph)

A **weighted (undirected) graph** is a triple which consists of

- a set V, elements of which we call **vertices** of the graph,
- a set $E \subset V \times V$, elements of which we call **edges** of the graph (an element $(u, v) \in E$ is called an edge, connecting the vertex u to the vertex v). In case of undirected graph, we identify the edge (u, v) with the edge (v, u);
- a set $W = \{\omega_{u,v} \,|(u, v) \in E\}$, elements of which we call **weights** of the graph (an element $\omega_{u,v} \in W$ is called the weight on the edge $(u, v) \in E$).

So the triple $G = (V, E, W)$ we call an (undirected) weighted graph.

Recall Some Definitions

Definition (Degree of a Vertex)

The **degree** of a vertex is the sum of the weights on edges incident on that vertex.

Definition (Path)

Two edges are **consecutive** if they have a common vertex. A **path** is a sequence of consecutive edges.

Definition (Circuit)

A **circuit** is a path which starts and ends at the same vertex.

Recall Some Definitions

Definition (Tree)

Two vertices are called **connected** if there is a path between them. A **connected graph** is a graph any two vertices of which are connected. A **tree** is a connected graph without circuits (more precisely, without circuits, which contains more than one edge).

Definition (Spanning Tree)

Let $G = (V, E, W)$ be a connected graph, a **spanning tree** is a tree with vertices V and edges from E.

Definition (Minimum Weight Spanning Tree)

The spanning tree with minimum sum of weights on its edges is called **minimum (weight) spanning tree**.

In a weighted graph there are efficient algorithms for computing minimum spanning trees.

Agglomerative Clustering with Graphs

Now let's describe an agglomerative clustering algorithm on graphs for image segmentation. The algorithm is originated from the paper *P. Felzenszwalb, D. Huttenlocher "Efficient Graph-Based Image Segmentation", 2003*.

Agglomerative Clustering with Graphs

Now let's describe an agglomerative clustering algorithm on graphs for image segmentation. The algorithm is originated from the paper *P. Felzenszwalb, D. Huttenlocher "Efficient Graph-Based Image Segmentation", 2003*.

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

- Each pixel is a vertex, i.e. $V = \{(x, y) \mid x = 0, \ldots, H-1,\ y = 0, \ldots, W-1\}$.

Agglomerative Clustering with Graphs

Now let's describe an agglomerative clustering algorithm on graphs for image segmentation. The algorithm is originated from the paper *P. Felzenszwalb, D. Huttenlocher "Efficient Graph-Based Image Segmentation", 2003*.

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

- Each pixel is a vertex, i.e. $V = \{(x, y) \mid x = 0, \ldots, H-1,\ y = 0, \ldots, W-1\}$.
- Each pixel is connected only to its neighbors (here we use 8-neighbors), i.e. for vertices $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ there is an edge $(v_1, v_2) \in E$ iff the pixel v_1 is a neighbor of v_2.

Agglomerative Clustering with Graphs

Now let's describe an agglomerative clustering algorithm on graphs for image segmentation. The algorithm is originated from the paper *P. Felzenszwalb, D. Huttenlocher "Efficient Graph-Based Image Segmentation", 2003*.

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

- Each pixel is a vertex, i.e. $V = \{(x, y) \mid x = 0, \ldots, H-1,\ y = 0, \ldots, W-1\}$.
- Each pixel is connected only to its neighbors (here we use 8-neighbors), i.e. for vertices $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ there is an edge $(v_1, v_2) \in E$ iff the pixel v_1 is a neighbor of v_2.
- The weight on the edge (v_1, v_2) between the pixels $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ is defined as $\omega_{v_1, v_2} = \|I(x_1, y_1) - I(x_2, y_2)\|$.

Agglomerative Clustering with Graphs

Note

As the weight ω_{v_1,v_2} is equal to the distance of pixels v_1 and v_2 in the *RGB* color-space, we interpret the weight $\omega_{u,v}$ as a measure of *dissimilarity* of the pixels u, v. In this interpretation we can assume that every two pixels (vertices of the graph) u and v are connected with an edge with the weight

$$\omega_{u,v} = \begin{cases} \omega_{u,v} & \text{if } (u,v) \in E \\ +\infty & \text{otherwise.} \end{cases}$$

So in this case we can identify the set Ω with a matrix of shape $|V| \times |V|$ (here $|V| = HW$): $\Omega = (\omega_{i,j})_{i,j=0}^{|V|}$. So we get a *symmetric* matrix Ω, which is called the **adjacency matrix** of the graph G.

Agglomerative Clustering with Graphs

Recall that for agglomerative clustering we need to specify the *distance* of two clusters and the *criterion* when to stop.

Agglomerative Clustering with Graphs

Recall that for agglomerative clustering we need to specify the *distance* of two clusters and the *criterion* when to stop.

Defining the Distance

The distance between the clusters $C_1 = \{v_{i_1}, \dots \ v_{i_m}\}$ and $C_2 = \{v_{j_1}, \dots, v_{j_k}\}$ is measured by the natural way as the distance of two sets of pixels (in the *RGB* color-space):

$$dist(C_1, C_2) = min\{\|I(v) - I(u)\| \ : \ v \in C_1, u \in C_2\}.$$

Agglomerative Clustering with Graphs

Recall that for agglomerative clustering we need to specify the *distance* of two clusters and the *criterion* when to stop.

Defining the Distance

The distance between the clusters $C_1 = \{v_{i_1}, \dots \ v_{i_m}\}$ and $C_2 = \{v_{j_1}, \dots, v_{j_k}\}$ is measured by the natural way as the distance of two sets of pixels (in the *RGB* color-space):

$$dist(C_1, C_2) = min\{\|I(v) - I(u)\| \ : \ v \in C_1, u \in C_2\}.$$

We also need to specify the *merging criterion* to know for each iteration merge or not the two closest clusters.

Agglomerative Clustering with Graphs

Merging Criterion

We define a criterion, which will guide us to merge or not two (closest) clusters C_1 and C_2. For this we define an *internal distance* of a cluster C by the following way:

$$int(C) = max\{\omega_{u,v} \ : \ (u,v) \in E(T(C))\},$$

where $T(C)$ is the minimum weighted spanning three of the subgraph with vertices C, and $E(T(C))$ is the set of edges of the tree $T(C)$.

Agglomerative Clustering with Graphs

Merging Criterion

We define a criterion, which will guide us to merge or not two (closest) clusters C_1 and C_2. For this we define an *internal distance* of a cluster C by the following way:

$$int(C) = max\{\omega_{u,v} \ : \ (u,v) \in E(T(C))\},$$

where $T(C)$ is the minimum weighted spanning three of the subgraph with vertices C, and $E(T(C))$ is the set of edges of the tree $T(C)$. Then we merge the clusters C_1 and C_2, if

$$dist(C_1, C_2) \leq min\{int(C_1) + \tau(C_1), int(C_2) + \tau(C_2)\},$$

where $\tau(C) = \frac{k}{|C|}$, and k is some constant.
We add the terms $\tau(C_1)$ and $\tau(C_2)$, since in early stages of the algorithm the internal distances of the clusters are 0 (if the cluster consists of one element) or are very small, so otherwise they will not be merged.

Agglomerative Clustering with Graphs

And finally, our *stop criterion* can be the following:

Stop Criterion

If the number of clusters at the k^{th} iteration is K, then we stop in the case when during iterations $k, k+1, \ldots, k+C_K^2$ we do not merge any two clusters.

Agglomerative Clustering with Graphs

And finally, our *stop criterion* can be the following:

Stop Criterion

If the number of clusters at the k^{th} iteration is K, then we stop in the case when during iterations $k, k+1, \ldots, k+C_K^2$ we do not merge any two clusters.

Note

Notice that here is also useful to get rid of noise in order to avoid fake small clusters. So before the clustering algorithm, we blur the image with Gaussian blur.

Agglomerative Clustering with Graphs

So, in summary, algorithm for agglomerative clustering with graphs in an image is the following:

The Algorithm

- Blur the image with Gaussian blur (with parameter $\sigma = 0.8$ as in some cases in the paper). Proceed with the blurred image.

Agglomerative Clustering with Graphs

So, in summary, algorithm for agglomerative clustering with graphs in an image is the following:

The Algorithm

- Blur the image with Gaussian blur (with parameter $\sigma = 0.8$ as in some cases in the paper). Proceed with the blurred image.
- Define the graph with its adjacency matrix.

Agglomerative Clustering with Graphs

So, in summary, algorithm for agglomerative clustering with graphs in an image is the following:

The Algorithm

- Blur the image with Gaussian blur (with parameter $\sigma = 0.8$ as in some cases in the paper). Proceed with the blurred image.
- Define the graph with its adjacency matrix.
- Start with each pixel as one cluster.

Agglomerative Clustering with Graphs

So, in summary, algorithm for agglomerative clustering with graphs in an image is the following:

The Algorithm

- Blur the image with Gaussian blur (with parameter $\sigma = 0.8$ as in some cases in the paper). Proceed with the blurred image.
- Define the graph with its adjacency matrix.
- Start with each pixel as one cluster.
- Begin iterations while the *Stop Criterion* is not satisfied.

Agglomerative Clustering with Graphs

So, in summary, algorithm for agglomerative clustering with graphs in an image is the following:

The Algorithm

- Blur the image with Gaussian blur (with parameter $\sigma = 0.8$ as in some cases in the paper). Proceed with the blurred image.
- Define the graph with its adjacency matrix.
- Start with each pixel as one cluster.
- Begin iterations while the *Stop Criterion* is not satisfied.
- for each iteration find the *closest* clusters and decide by the *merging criterion* to merge them or not, and in case of not merging do not consider them as the closest clusters in the next iteration.

Overview

Watershed Segmentation

The watershed algorithm for image segmentation is originated from the paper *Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In Mathematical Morphology in Image Processing (Ed. E. R. Dougherty), pages 433–481 (1993)*.

Watershed Segmentation

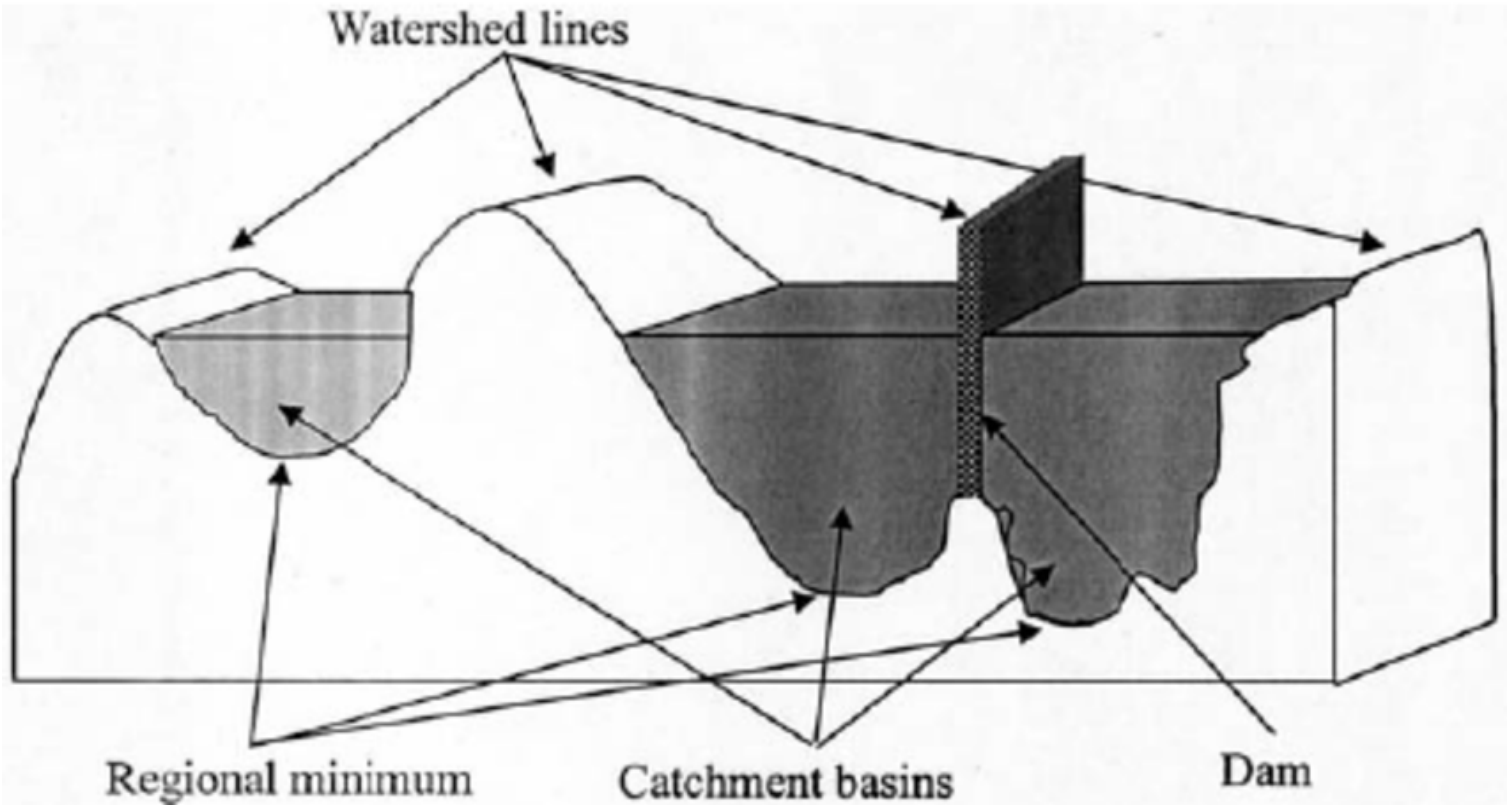
The watershed algorithm for image segmentation is originated from the paper *Serge Beucher and Fernand Meyer. The morphological approach to segmentation: the watershed transformation. In Mathematical Morphology in Image Processing (Ed. E. R. Dougherty), pages 433–481 (1993).*

The Idea of Watershed

Let $f : [0,1]^2 \to [0,1]$ be an image and let us consider the following surface: $S = \{(x, y, z) \in \mathbb{R} \ : \ z = f(x, y)\}$. We know that S have some peaks and troughs. Now imagine that holes are pierced in local minima of f and the level of water is rising from the 0-level. At the beginning levels of water we can clearly distinguish some "basins" partially filled with water the level of which is constantly rising. In some level the waters of some "basins" will be merged. Our goal is to detect this moments and put the watershed dams in order to avoid water mixing. So in this way we will get the "basins", the set of which will be the segmentation result.

Watershed Algorithm

The following image is illustrating the idea of watershed segmentation algorithm.



Watershed Algorithm

Now let us describe an algorithm which simulates the rising water and segmenting the "basins".

Watershed Algorithm

Now let us describe an algorithm which simulates the rising water and segmenting the "basins".

The Watershed Algorithm

Let $I \in \mathbb{R}^{H \times W}$ be a grayscaled image with integer values from $\{0, 1, \ldots, 255\}$.

- Find local minima of I and mark them with different labels.

Watershed Algorithm

Now let us describe an algorithm which simulates the rising water and segmenting the "basins".

The Watershed Algorithm

Let $I \in \mathbb{R}^{H \times W}$ be a grayscaled image with integer values from $\{0, 1, \ldots, 255\}$.

- Find local minima of I and mark them with different labels.
- Make a dictionary D with keys as (levels of water) $0, 1, \ldots 255$, and value of a key $h \in \{0, \ldots 255\}$ define as the set of local minima which have the value h. It is obvious that for some keys their values can be empty sets.

Watershed Algorithm

The Watershed Algorithm

- Repeat the following until the above-mentioned dictionary D has empty values:

  - 1) among the keys k of D, for which the set $D[k]$ is not empty, take the smallest k and consider the set $S = D[k]$;

The Watershed Algorithm

- Repeat the following until the above-mentioned dictionary D has empty values:

  - 1) among the keys k of D, for which the set $D[k]$ is not empty, take the smallest k and consider the set $S = D[k]$;

  - 2) for every element (i, j) in this set S mark unlabeled neighbors of (i, j) (i.e. unlabeled elements of the set $\{(i + \varepsilon, j + \varepsilon) \ : \ \varepsilon = -1, 0, 1\}$) with the label of (i, j), denote the set of new marked indices by $\tilde{S}$;

Watershed Algorithm

The Watershed Algorithm

- Repeat the following until the above-mentioned dictionary D has empty values:

  - 1) among the keys k of D, for which the set $D[k]$ is not empty, take the smallest k and consider the set $S = D[k]$;

  - 2) for every element (i, j) in this set S mark unlabeled neighbors of (i, j) (i.e. unlabeled elements of the set $\{(i + \varepsilon, j + \varepsilon) \ : \ \varepsilon = -1, 0, 1\}$) with the label of (i, j), denote the set of new marked indices by $\tilde{S}$;

  - 3) add the elements of $\tilde{S}$ to the dictionary D: for all elements $(u, v) \in \tilde{S}$ add (u, v) to the set $D[I_{u,v}]$;

Watershed Algorithm

The Watershed Algorithm

- Repeat the following until the above-mentioned dictionary D has empty values:

  - 1) among the keys k of D, for which the set $D[k]$ is not empty, take the smallest k and consider the set $S = D[k]$;

  - 2) for every element (i, j) in this set S mark unlabeled neighbors of (i, j) (i.e. unlabeled elements of the set $\{(i + \varepsilon, j + \varepsilon) \ : \ \varepsilon = -1, 0, 1\}$) with the label of (i, j), denote the set of new marked indices by $\tilde{S}$;

  - 3) add the elements of $\tilde{S}$ to the dictionary D: for all elements $(u, v) \in \tilde{S}$ add (u, v) to the set $D[I_{u,v}]$;

  - 4) remember the neighbor indices in which pixels are marked by a different from (i, j) pixel's label; these pixels will form **the watershed lines**;

Watershed Algorithm

The Watershed Algorithm

- Repeat the following until the above-mentioned dictionary D has empty values:

  - 1) among the keys k of D, for which the set $D[k]$ is not empty, take the smallest k and consider the set $S = D[k]$;

  - 2) for every element (i, j) in this set S mark unlabeled neighbors of (i, j) (i.e. unlabeled elements of the set $\{(i + \varepsilon, j + \varepsilon) \ : \ \varepsilon = -1, 0, 1\}$) with the label of (i, j), denote the set of new marked indices by $\tilde{S}$;

  - 3) add the elements of $\tilde{S}$ to the dictionary D: for all elements $(u, v) \in \tilde{S}$ add (u, v) to the set $D[I_{u,v}]$;

  - 4) remember the neighbor indices in which pixels are marked by a different from (i, j) pixel's label; these pixels will form **the watershed lines**;

  - 5) remove the elements of the set S from the updated set $D[k]$.

Watershed Algorithm: Oversegmentation

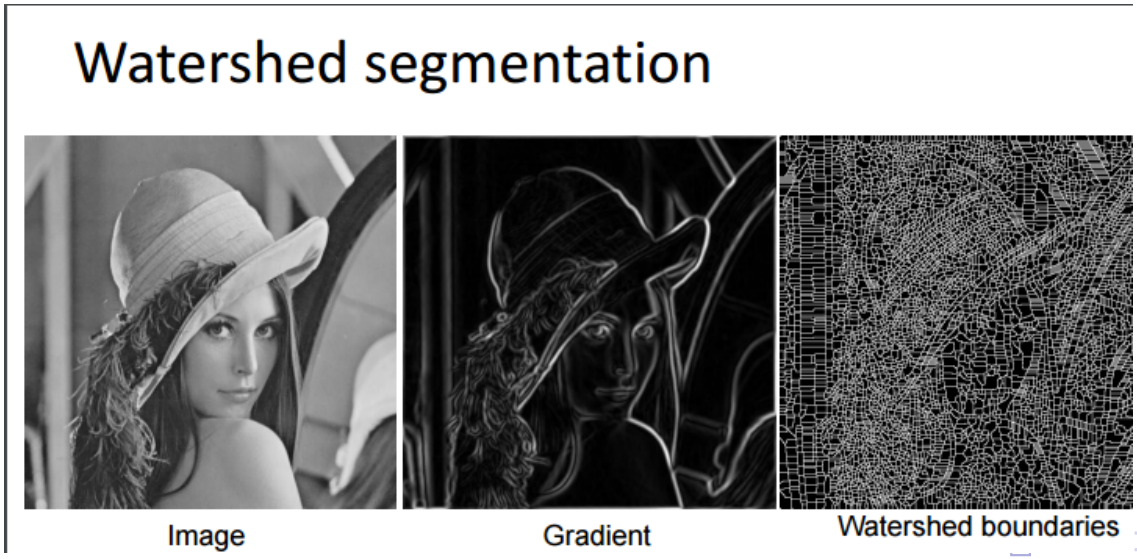
Let us notice that according to the above-mentioned algorithm, we get all pixels labeled with some label.

Watershed Algorithm: Oversegmentation

Let us notice that according to the above-mentioned algorithm, we get all pixels labeled with some label.

Note

Note also, that if we consider every local minimum as labeled index in the first step of the algorithm, we can end up with many segments (the number of segments is the number of local minima). This phenomenon is called **oversegmentation**. You can see the oversegmentation with watershed algorithm on the image below.



Watershed Algorithm: Markers

As it is noticed earlier, the reason of oversegmentation is the number of local minima. But not all local minima are significant. So, in order to avoid oversegmentation, we introduce the concept of *markers*. Markers are nothing else than predefined pixels which will be treated as only marked local minima. This approach is called **marked watershed**. So instead of labeling (and including in values of D) all local minima, now we label (and include in values of D) only predefined markers.

Watershed Algorithm: Markers

As it is noticed earlier, the reason of oversegmentation is the number of local minima. But not all local minima are significant. So, in order to avoid oversegmentation, we introduce the concept of *markers*. Markers are nothing else than predefined pixels which will be treated as only marked local minima. This approach is called **marked watershed**. So instead of labeling (and including in values of D) all local minima, now we label (and include in values of D) only predefined markers.

Marked Watershed Algorithm

- By the watershed algorithm, initially marked only in marker indices, obtain the watershed lines.

Watershed Algorithm: Markers

As it is noticed earlier, the reason of oversegmentation is the number of local minima. But not all local minima are significant. So, in order to avoid oversegmentation, we introduce the concept of *markers*. Markers are nothing else than predefined pixels which will be treated as only marked local minima. This approach is called **marked watershed**. So instead of labeling (and including in values of D) all local minima, now we label (and include in values of D) only predefined markers.

Marked Watershed Algorithm

- By the watershed algorithm, initially marked only in marker indices, obtain the watershed lines.
- Add the obtained watershed lines as markers to the initial markers. Obtained watershed lines are marked with the same label which correspond to the background.

Watershed Algorithm: Markers

As it is noticed earlier, the reason of oversegmentation is the number of local minima. But not all local minima are significant. So, in order to avoid oversegmentation, we introduce the concept of *markers*. Markers are nothing else than predefined pixels which will be treated as only marked local minima. This approach is called **marked watershed**. So instead of labeling (and including in values of D) all local minima, now we label (and include in values of D) only predefined markers.

Marked Watershed Algorithm

- By the watershed algorithm, initially marked only in marker indices, obtain the watershed lines.
- Add the obtained watershed lines as markers to the initial markers. Obtained watershed lines are marked with the same label which correspond to the background.
- Use updated markers and watershed algorithm to obtain the final segmentation result.

Overview

Normalized Cut Algorithm

Normalized Cut algorithm for image segmentation [3] is a clustering algorithm on graph, which belongs to the *divisive* clustering algorithms. As in divisive clustering methods the *distance* between clusters is a key concept, we will start from defining this distance. Then, based on this concept of distance, we will partition a big cluster C into two small clusters C_1, C_2, in such way that $dist(C_1, C_2)$ is the highest among all partitions of C into two small parts (clusters).

[3] Described in the paper: J. Shi, J. Malik, "Normalized cuts and image segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8):888-905, August 2000.

Normalized Cut Algorithm

Normalized Cut algorithm for image segmentation [3] is a clustering algorithm on graph, which belongs to the *divisive* clustering algorithms. As in divisive clustering methods the *distance* between clusters is a key concept, we will start from defining this distance. Then, based on this concept of distance, we will partition a big cluster C into two small clusters C_1, C_2, in such way that $dist(C_1, C_2)$ is the highest among all partitions of C into two small parts (clusters).

Note

In other words, as in the first step of divisive clustering we have the whole image as one big cluster, we start from description of a method (which is essentially the *Normalized Cut Algorithm*) for segmenting an image into two regions.

[3] Described in the paper: J. Shi, J. Malik, "Normalized cuts and image segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8):888-905, August 2000.

Normalized Cut Algorithm: The Graph Description

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

Normalized Cut Algorithm: The Graph Description

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

- Each pixel is a vertex, i.e.

$$V = \{(x, y) \ : \ x = 0, \ldots, H-1, \ y = 0, \ldots, W-1\}.$$

Normalized Cut Algorithm: The Graph Description

Defining the Graph

Let $I \in \mathbb{R}^{H \times W \times 3}$ be a colored (3-channeled) image. We define a graph $G = (V, E, \Omega)$ by the following way.

- Each pixel is a vertex, i.e.

$$V = \{(x, y) \ : \ x = 0, \ldots, H-1, \ y = 0, \ldots, W-1\}.$$

- Every two pixels are connected to each other, i.e. $E = V \times V$.

Normalized Cut Algorithm: The Graph Description

Defining the Graph

- The weight ω_{v_1,v_2} on the edge (v_1, v_2) is showing the *similarity* of vertices (pixels indexed by the vertices) v_1, v_2. So any measure, describing a similarity between pixels can be taken as ω_{v_1,v_2}.

Normalized Cut Algorithm: The Graph Description

Defining the Graph

- The weight ω_{v_1,v_2} on the edge (v_1, v_2) is showing the *similarity* of vertices (pixels indexed by the vertices) v_1, v_2. So any measure, describing a similarity between pixels can be taken as ω_{v_1,v_2}. For clarity we take (as in paper)

$$\omega_{i,j} = e^{-\frac{\|F_i - F_j\|^2}{\sigma_I^2}} \cdot \begin{cases} e^{-\frac{\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if } \|X_i - X_j\|^2 \leq r \\ 0 & \text{otherwise} \end{cases},$$

where

Normalized Cut Algorithm: The Graph Description

Defining the Graph

- The weight ω_{v_1,v_2} on the edge (v_1, v_2) is showing the *similarity* of vertices (pixels indexed by the vertices) v_1, v_2. So any measure, describing a similarity between pixels can be taken as ω_{v_1,v_2}. For clarity we take (as in paper)

$$\omega_{i,j} = e^{-\frac{\|F_i - F_j\|^2}{\sigma_I^2}} \cdot \begin{cases} e^{-\frac{\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if } \|X_i - X_j\|^2 \leq r \\ 0 & \text{otherwise} \end{cases},$$

where

  - F_i is a feature vector describing the pixel i (color, texture, orientation, etc.)

Normalized Cut Algorithm: The Graph Description

Defining the Graph

- The weight ω_{v_1,v_2} on the edge (v_1, v_2) is showing the *similarity* of vertices (pixels indexed by the vertices) v_1, v_2. So any measure, describing a similarity between pixels can be taken as ω_{v_1,v_2}. For clarity we take (as in paper)

$$\omega_{i,j} = e^{-\frac{\|F_i - F_j\|^2}{\sigma_I^2}} \cdot \begin{cases} e^{-\frac{\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if } \|X_i - X_j\|^2 \leq r \\ 0 & \text{otherwise} \end{cases},$$

where

  - F_i is a feature vector describing the pixel i (color, texture, orientation, etc.)
  - X_i is the $2d$ spatial coordinates of the pixel i

Normalized Cut Algorithm: The Graph Description

Defining the Graph

- The weight ω_{v_1,v_2} on the edge (v_1, v_2) is showing the *similarity* of vertices (pixels indexed by the vertices) v_1, v_2. So any measure, describing a similarity between pixels can be taken as ω_{v_1,v_2}. For clarity we take (as in paper)

$$\omega_{i,j} = e^{-\frac{\|F_i - F_j\|^2}{\sigma_I^2}} \cdot \begin{cases} e^{-\frac{\|X_i - X_j\|^2}{\sigma_X^2}} & \text{if } \|X_i - X_j\|^2 \leq r \\ 0 & \text{otherwise} \end{cases},$$

where

  - F_i is a feature vector describing the pixel i (color, texture, orientation, etc.)
  - X_i is the $2d$ spatial coordinates of the pixel i
  - σ_I, σ_X and r are some predefined parameters

Normalized Cut Algorithm: The Distance of Subclusters

Note

By the above-mentioned definition of the graph, here we also have an adjacency matrix, which we will identify with the set Ω, i.e. $\Omega = (\omega_{i,j})_{i,j=1}^{|V|}$ (recall that $|V|$ is the number of pixels, i.e. $|V| = HW$).

Normalized Cut Algorithm: The Distance of Subclusters

Note

By the above-mentioned definition of the graph, here we also have an adjacency matrix, which we will identify with the set Ω, i.e. $\Omega = (\omega_{i,j})_{i,j=1}^{|V|}$ (recall that $|V|$ is the number of pixels, i.e. $|V| = HW$).

Now let us describe a method to partition the cluster $C = I$ into two small clusters $C_1, C_2 \subset C$.

Normalized Cut Algorithm: The Distance of Subclusters

Note

By the above-mentioned definition of the graph, here we also have an adjacency matrix, which we will identify with the set Ω, i.e. $\Omega = (\omega_{i,j})_{i,j=1}^{|V|}$ (recall that $|V|$ is the number of pixels, i.e. $|V| = HW$).

Now let us describe a method to partition the cluster $C = I$ into two small clusters $C_1, C_2 \subset C$. For any partition $\{C_1, C_2\}$ we define a **cut** (the intuition behind this term is that for partitioning the *graph*, we need to *cut* some edges of it, i.e. in our case to give 0 values to some weights, namely the weights connecting the vertices from C_1 to the vertices from C_2):

$$Cut(C_1, C_2) = \sum_{i \in C_1,\, j \in C_2} \omega_{i,j}.$$

We can interpret this as the *amount* we need to *"pay"* for *cutting* the edges between C_1 and C_2.

Normalized Cut Algorithm: The Distance of Subclusters

Note

Normalized Cut Algorithm: The Distance of Subclusters

Note

If the $Cut(C_1, C_2)$ is large, it means that we need to make much effort for separating the subsets C_1 and C_2 (i.e. for partitioning C into C_1 and C_2), so in some sense C_1 and C_2 are *close* to each other.

Normalized Cut Algorithm: The Distance of Subclusters

Note

If the $Cut(C_1, C_2)$ is large, it means that we need to make much effort for separating the subsets C_1 and C_2 (i.e. for partitioning C into C_1 and C_2), so in some sense C_1 and C_2 are *close* to each other.

It leads us to the concept of a distance between two clusters C_1, C_2 as the inverse quantity of cut:

$$distance(C_1, C_2) = \frac{1}{Cut(C_1, C_2)}.$$

Normalized Cut Algorithm: The Distance of Subclusters

Note

If the $Cut(C_1, C_2)$ is large, it means that we need to make much effort for separating the subsets C_1 and C_2 (i.e. for partitioning C into C_1 and C_2), so in some sense C_1 and C_2 are *close* to each other.

It leads us to the concept of a distance between two clusters C_1, C_2 as the inverse quantity of cut:

$$distance(C_1, C_2) = \frac{1}{Cut(C_1, C_2)}.$$

So, as described in divisive clustering approach, we need to find such partition $\{A, B\}$ of the cluster C, that:

$$distance(A, B) = \max_{\{C_1, C_2\} \text{ is a partition of } C} \{distance(C_1, C_2)\}.$$

Normalized Cut Algorithm: The Distance of Subclusters

Note

If the $Cut(C_1, C_2)$ is large, it means that we need to make much effort for separating the subsets C_1 and C_2 (i.e. for partitioning C into C_1 and C_2), so in some sense C_1 and C_2 are *close* to each other.

It leads us to the concept of a distance between two clusters C_1, C_2 as the inverse quantity of cut:

$$distance(C_1, C_2) = \frac{1}{Cut(C_1, C_2)}.$$

So, as described in divisive clustering approach, we need to find such partition $\{A, B\}$ of the cluster C, that:

$$distance(A, B) = \max_{\{C_1, C_2\} \text{ is a partition of } C} \{distance(C_1, C_2)\}.$$

But this is the same as

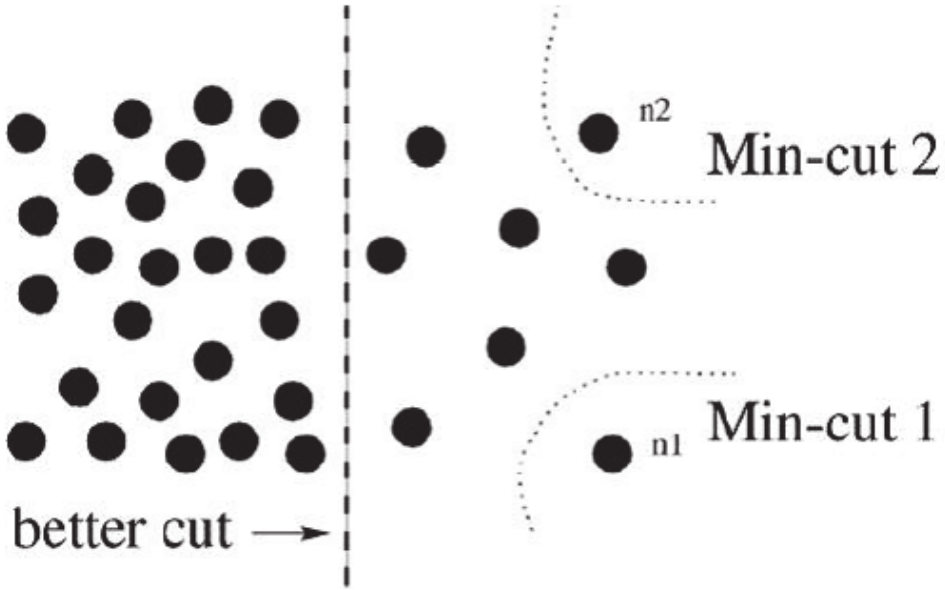
$$Cut(A, B) = \min_{\{C_1, C_2\} \text{ is a partition of } C} \{Cut(C_1, C_2)\}.$$

Normalized Cut Algorithm: The Criterion for Partitioning

So, we need to find the smallest cut among the all partitions of the cluster C and take the corresponding partition.

Normalized Cut Algorithm: The Criterion for Partitioning

So, we need to find the smallest cut among the all partitions of the cluster C and take the corresponding partition.
Although minimizing the cut between partitions can seem very intuitively, sometimes it leads to unwanted partitions. You can see this phenomenon on the following image (here vertices are shown as points in $2d$ feature-space, and weights between them are shown as distances between them (it can be misleading compared with our case, since we treat weights as similarities instead of dissimilarities)).



Normalized Cut Algorithm: The Normalized Cut

So instead of minimizing *the cut* of a partition, we minimize another quantity, called the **normalized cut** of the partition, which is defined by the following way.

Normalized Cut Algorithm: The Normalized Cut

So instead of minimizing *the cut* of a partition, we minimize another quantity, called the **normalized cut** of the partition, which is defined by the following way.
At first we define the similar notion to the cut: **association**:

$$Assoc(A, B) = \sum_{i \in A, j \in B} \omega_{i,j}, \text{ for any subsets } A, B \subset C.$$

Normalized Cut Algorithm: The Normalized Cut

So instead of minimizing *the cut* of a partition, we minimize another quantity, called the **normalized cut** of the partition, which is defined by the following way.
At first we define the similar notion to the cut: **association**:

$$Assoc(A, B) = \sum_{i \in A, j \in B} \omega_{i,j}, \text{ for any subsets } A, B \subset C.$$

Note, that the difference between *cut* and *association* is that the *cut* is defined only for a *partition* of C (i.e. $A, B \subset C, A \cap B = \emptyset, A \cup B = C$).

Normalized Cut Algorithm: The Normalized Cut

So instead of minimizing *the cut* of a partition, we minimize another quantity, called the **normalized cut** of the partition, which is defined by the following way.
At first we define the similar notion to the cut: **association**:

$$Assoc(A, B) = \sum_{i \in A, j \in B} \omega_{i,j}, \text{ for any subsets } A, B \subset C.$$

Note, that the difference between *cut* and *association* is that the *cut* is defined only for a *partition* of C (i.e. $A, B \subset C, A \cap B = \emptyset, A \cup B = C$).

Definition (Normalized Cut)

The **normalized cut** of a partition $\{A, B\}$ of the cluster C is

$$NCut(A, B) = \frac{Cut(A, B)}{Assoc(A, C)} + \frac{Cut(A, B)}{Assoc(B, C)}.$$

Normalized Cut Algorithm: The Associated Vector

So, for partitioning the cluster C into two subclusters, we need to find the partition with minimal normalized cut.

Normalized Cut Algorithm: The Associated Vector

So, for partitioning the cluster C into two subclusters, we need to find the partition with minimal normalized cut.

The associated vector

Let $\{A, B\}$ be a partition of C. We can associate to this partition a vector $\mathbf{x}(A, B) = \mathbf{x} \in \{-1, 1\}^{|C|}$, such that for every $i = 0, 1, \ldots, |C| - 1$

$$\mathbf{x}_i = \begin{cases} 1 & \text{if the } i^{th} \text{ vertex (pixel) of } C \text{ belongs to } A \\ -1 & \text{if the } i^{th} \text{ vertex (pixel) of } C \text{ belongs to } B \end{cases}$$

Normalized Cut Algorithm: The Associated Vector

So, for partitioning the cluster C into two subclusters, we need to find the partition with minimal normalized cut.

The associated vector

Let $\{A, B\}$ be a partition of C. We can associate to this partition a vector $\mathbf{x}(A, B) = \mathbf{x} \in \{-1, 1\}^{|C|}$, such that for every $i = 0, 1, \ldots, |C| - 1$

$$\mathbf{x}_i = \begin{cases} 1 & \text{if the } i^{th} \text{ vertex (pixel) of } C \text{ belongs to } A \\ -1 & \text{if the } i^{th} \text{ vertex (pixel) of } C \text{ belongs to } B \end{cases}$$

So we can write the normalized cut as

$$NCut(A, B) = \frac{\sum_{\mathbf{x}_i > 0, \mathbf{x}_j < 0} -\omega_{i,j} \mathbf{x}_i \mathbf{x}_j}{\sum_{x_i > 0} \sum_j \omega_{i,j}} + \frac{\sum_{\mathbf{x}_i < 0, \mathbf{x}_j > 0} -\omega_{i,j} \mathbf{x}_i \mathbf{x}_j}{\sum_{x_j < 0} \sum_i \omega_{i,j}}$$

Normalized Cut Algorithm: The Associated Vector

Hence finding the partition with minimal normalized cut is equivalent to the solution of the following problem.

Normalized Cut Algorithm: The Associated Vector

Hence finding the partition with minimal normalized cut is equivalent to the solution of the following problem.

Reformulation in Terms of the Associated Vector

Find a vector $\mathbf{x} \in \{-1, 1\}^{|C|}$, such that

$$NCut(\mathbf{x}) = \frac{\sum_{\mathbf{x}_i>0, \mathbf{x}_j<0} -\omega_{i,j}\mathbf{x}_i\mathbf{x}_j}{\sum_{x_i>0}\sum_j \omega_{i,j}} + \frac{\sum_{\mathbf{x}_i<0, \mathbf{x}_j>0} -\omega_{i,j}\mathbf{x}_i\mathbf{x}_j}{\sum_{x_j<0}\sum_i \omega_{i,j}}$$

be minimal.

Normalized Cut Algorithm: The Associated Vector

Hence finding the partition with minimal normalized cut is equivalent to the solution of the following problem.

Reformulation in Terms of the Associated Vector

Find a vector $\mathbf{x} \in \{-1, 1\}^{|C|}$, such that

$$NCut(\mathbf{x}) = \frac{\sum_{\mathbf{x}_i>0, \mathbf{x}_j<0} -\omega_{i,j}\mathbf{x}_i\mathbf{x}_j}{\sum_{x_i>0}\sum_j \omega_{i,j}} + \frac{\sum_{\mathbf{x}_i<0, \mathbf{x}_j>0} -\omega_{i,j}\mathbf{x}_i\mathbf{x}_j}{\sum_{x_j<0}\sum_i \omega_{i,j}}$$

be minimal.

The main problem of solving this kind of problems is the big number of candidates for the vector $\mathbf{x}$, which makes infeasible to solve it by brute force. Hence we introduce some "relaxations" to this problem. But at first we modify the objective we want to minimize.

Normalized Cut Algorithm: Some Notations

For this let us make some notations.

- Denote by d_i the degree of the i^{th} vertex of C:

$$d_i = \sum_j \omega_{i,j}.$$

- Denote by D the diagonal matrix with entries d_i:

$$D = diag(d_0, d_1, \ldots, d_{|C|-1}).$$

- Denote by L the **graph laplacian**:

$$L = D - W.$$

- Denote by $\mathcal{L}$ the **normalized graph laplacian**:

$$\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}.$$

Normalized Cut Algorithm: The Reformulation

It can be shown that in above-mentioned notations the problem $\min_{\mathbf{x}} NCut(x)$ is equivalent to the problem

$$\begin{cases} \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \\ \mathbf{y} \in \{1, -b\}^{|C|}, \quad \text{where } b = \frac{\sum_{\mathbf{x}_i>0} d_i}{\sum_{\mathbf{x}_i<0} d_i} \\ \mathbf{y}^T D\mathbf{1} = \mathbf{0}, \quad \text{where } \mathbf{1} \text{ and } \mathbf{0} \text{ are vectors of all 0 and 1 respectively} \end{cases}.$$

Normalized Cut Algorithm: The Reformulation

It can be shown that in above-mentioned notations the problem $\min_{\mathbf{x}} NCut(x)$ is equivalent to the problem

$$\begin{cases} \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \\ \mathbf{y} \in \{1, -b\}^{|C|}, \quad \text{where } b = \frac{\sum_{\mathbf{x}_i>0} d_i}{\sum_{\mathbf{x}_i<0} d_i} \\ \mathbf{y}^T D\mathbf{1} = \mathbf{0}, \quad \text{where } \mathbf{1} \text{ and } \mathbf{0} \text{ are vectors of all 0 and 1 respectively} \end{cases}.$$

In this reformulation of the problem, we use

$$\mathbf{y} = \frac{(1+\mathbf{x}) - b(1-\mathbf{x})}{2}.$$

Normalized Cut Algorithm: The Reformulation

It can be shown that in above-mentioned notations the problem $\min_{\mathbf{x}} NCut(x)$ is equivalent to the problem

$$\begin{cases} \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \\ \mathbf{y} \in \{1, -b\}^{|C|}, \quad \text{where } b = \frac{\sum_{\mathbf{x}_i>0} d_i}{\sum_{\mathbf{x}_i<0} d_i} \\ \mathbf{y}^T D\mathbf{1} = \mathbf{0}, \quad \text{where } \mathbf{1} \text{ and } \mathbf{0} \text{ are vectors of all 0 and 1 respectively} \end{cases}.$$

In this reformulation of the problem, we use

$$\mathbf{y} = \frac{(1+\mathbf{x}) - b(1-\mathbf{x})}{2}.$$

So if we have a solution $\mathbf{y}$ of above-mentioned problem (imagine the b is given), we can get $\mathbf{x} = \frac{2\mathbf{y}+b-1}{1+b}$.

Normalized Cut Algorithm: The Relaxation

Note

Essentially we do not need to know the exact value of b to get $\mathbf{x}$ from $\mathbf{y}$. Indeed, we can just take $\mathbf{x}_i = sign(\mathbf{y}_i)$.

Normalized Cut Algorithm: The Relaxation

Note

Essentially we do not need to know the exact value of b to get $\mathbf{x}$ from $\mathbf{y}$. Indeed, we can just take $\mathbf{x}_i = sign(\mathbf{y}_i)$.

Note

As we have mentioned earlier, the number of candidates for the solution of above-mentioned problem is $2^{|C|}$ (in case, when the b is given). So it becomes infeasible to directly solve it. Thus we introduce some "relaxation" on this problem, by omitting the condition $\mathbf{y} \in \{1, -b\}^{|C|}$ (and also notice that we get rid of b). After relaxation we get the following problem:

$$\begin{cases} \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}} \\ \mathbf{y}^T D\mathbf{1} = \mathbf{0} \end{cases}.$$

Normalized Cut Algorithm: The Solution of Relaxation

Recall

It is well known fact, that the solution of the above-mentioned problem can be obtained from the *second eigenvector of the normalized graph laplacian* $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.

By *second eigenvector* we mean the eigenvector corresponding to the second smallest eigenvalue and orthogonal to the eigenvector corresponding to the first smallest eigenvalue.

Normalized Cut Algorithm: The Solution of Relaxation

Recall

It is well known fact, that the solution of the above-mentioned problem can be obtained from the *second eigenvector of the normalized graph laplacian* $\mathcal{L} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$.
By *second eigenvector* we mean the eigenvector corresponding to the second smallest eigenvalue and orthogonal to the eigenvector corresponding to the first smallest eigenvalue.

The Solution

Let $\mathbf{z}$ be the second eigenvalue of the normalized graph laplacian. Then it can be shown that the vector $\mathbf{y} = D^{-\frac{1}{2}}\mathbf{z}$ is the solution of the *relaxed problem*.

Normalized Cut Algorithm: The Solution of the Initial Problem

Now when we have a solution $\mathbf{y}$ of the relaxed problem, we can take some threshold value t and obtain the vector $\tilde{\mathbf{y}}(t)$:

$$\tilde{\mathbf{y}}(t)_i = \begin{cases} 1 & \text{if } \mathbf{y}_i \geq t \\ -1 & \text{otherwise} \end{cases}.$$

After, we can take also $\mathbf{x} = sign(\tilde{\mathbf{y}}(t)) = \tilde{\mathbf{y}}(t)$.

Normalized Cut Algorithm: The Solution of the Initial Problem

Now when we have a solution $\mathbf{y}$ of the relaxed problem, we can take some threshold value t and obtain the vector $\tilde{\mathbf{y}}(t)$:

$$\tilde{\mathbf{y}}(t)_i = \begin{cases} 1 & \text{if } \mathbf{y}_i \geq t \\ -1 & \text{otherwise} \end{cases}.$$

After, we can take also $\mathbf{x} = sign(\tilde{\mathbf{y}}(t)) = \tilde{\mathbf{y}}(t)$.

Question

Here the question arises: How to choose an appropriate threshold value t?

Normalized Cut Algorithm: The Solution of the Initial Problem

Now when we have a solution $\mathbf{y}$ of the relaxed problem, we can take some threshold value t and obtain the vector $\tilde{\mathbf{y}}(t)$:

$$\tilde{\mathbf{y}}(t)_i = \begin{cases} 1 & \text{if } \mathbf{y}_i \geq t \\ -1 & \text{otherwise} \end{cases}.$$

After, we can take also $\mathbf{x} = sign(\tilde{\mathbf{y}}(t)) = \tilde{\mathbf{y}}(t)$.

Question

Here the question arises: How to choose an appropriate threshold value t?

Answer

For defining the optimal threshold value t, we try all possible thresholds among the values $\mathbf{y}_0, \dots \mathbf{y}_{|C|-1}$, and take the one for which the corresponding normalized cut $NCut(\mathbf{x}) = NCut(\tilde{\mathbf{y}}(t))$ is minimal.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

- Obtain the normalized graph laplacian of the cluster C.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

- Obtain the normalized graph laplacian of the cluster C.
- Obtain the second eigenvector $\mathbf{z}$ of this normalized graph laplacian.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

- Obtain the normalized graph laplacian of the cluster C.
- Obtain the second eigenvector $\mathbf{z}$ of this normalized graph laplacian.
- Compute $\mathbf{y} = D^{-\frac{1}{2}}\mathbf{z}$.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

- Obtain the normalized graph laplacian of the cluster C.
- Obtain the second eigenvector $\mathbf{z}$ of this normalized graph laplacian.
- Compute $\mathbf{y} = D^{-\frac{1}{2}}\mathbf{z}$.
- Find the optimal threshold value t_{op} and obtain the vector $\mathbf{x} = \tilde{\mathbf{y}}(t_{op})$.

Normalized Cut Algorithm: The Solution of the Initial Problem

Summary

So in summary, for getting the best partition of a cluster C into two small subclusters A and B, we need the following steps.

- Obtain the normalized graph laplacian of the cluster C.
- Obtain the second eigenvector $\mathbf{z}$ of this normalized graph laplacian.
- Compute $\mathbf{y} = D^{-\frac{1}{2}}\mathbf{z}$.
- Find the optimal threshold value t_{op} and obtain the vector $\mathbf{x} = \tilde{\mathbf{y}}(t_{op})$.
- Partition the cluster C into A and B such that $v_i \in A$ iff $\mathbf{x}_i = 1$, where v_i is the i^{th} vertex in the cluster C.

Normalized Cut Algorithm: The Stop Criterion

As we know, for divisive clustering algorithms, we need also a *criterion* according to which we will decide split a cluster or not.

Normalized Cut Algorithm: The Stop Criterion

As we know, for divisive clustering algorithms, we need also a *criterion* according to which we will decide split a cluster or not.

A Stop Criterion

We can simply stop partitioning the cluster C if its minimum normalized cut exceeds some predefined value.

Introduction to Computer Vision: Segmentation by Clustering

Navasardyan Shant

October 27, 2019