Efficient Graph Based

Image Segmentation

Darshan Makwana

Dec 2022

Contents

- Overview
- The Algorithm
- Running time of the Algorithm
- Implementations
- Closing Thoughts

Original Paper: http://people.cs.uchicago.edu/~pff/papers/seg-ijcv.pdf

Overview

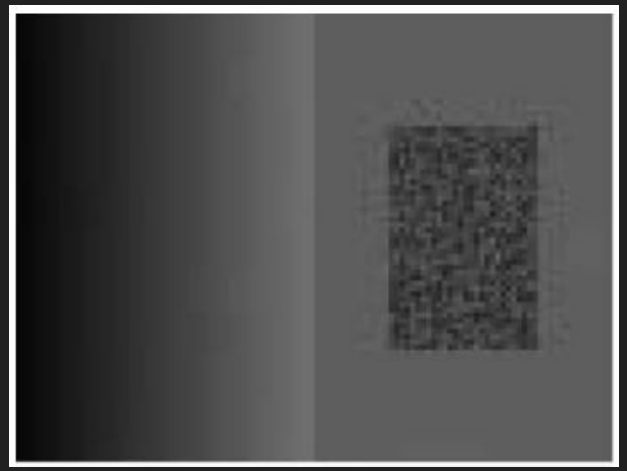
Rephrasing the problem of Image Segmentation in terms of a graph

- An Undirected Graph G=(V, E) where each $v_i \in V$ corresponds to a pixel in the image and each edge $(v_i,v_j) \in E$ has an weight w: $E \rightarrow R$ associated with it

- The weight is a non negative measure of dissimilarity which can be measured with location, intensity or local attributes

- The smaller the weight the greater the similarity between two pixels and they can be represented in the same segmentation

Intuition behind graph based approach



Regions cannot be obtained using purely local phenomenons

Widely varying intensity regions is not a sufficient evidence for boundary region

Graph based method measures evidence of boundary between two regions by measuring

- **Differences across boundaries**
- **Differences across neighbouring pixels in each region**

The Algorithm

- Constructing the Graph
- Finding Segmentation
- A Greedy Algorithm
- Existence of a Segmentation

Constructing the Graph

We smoothen out the image with a gaussian blur (σ=0.8) to compensate for the digitization effects
For Colored Images we apply the image segmentation for each channel

Grid Graphs:

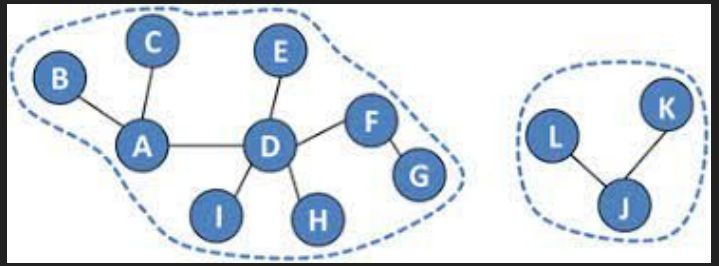
- Edges are connected in an 4 - connected sense
- Weights are taken as the absolute difference of pixel intensity

Nearest Neighbour Graphs:

- Edges are connected between all the points within some fixed distance d
- Weights calculated using euclidean distance between two pixel vertices

Finding Segmentation

The Evidence (D) that a boundary exists between two strongly connected (forest tree) components in a segmentation (S) is measured using dissimilarity of pixels



Measure of Dissimilarity along the boundary:

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w(v_i, v_j).$$

Measure of Dissimilarity within each component :

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

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)).$$

A Greedy Algorithm

0. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
1. Start with a segmentation S^0, where each vertex v_i is in its own component.
2. Repeat step 3 for $q = 1, \ldots, m$.
3. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_a) \leq MInt(C_i^{q-1}, C_i^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
4. Return $S = S^m$.

- At each step of the algorithm we choose the edge that has the minimum weight connecting any two component and check for the predicate
- The Reason being it is more likely for the predicate being true for edges with smaller weights that with larger weights
- So we sort the edges according to their weights and check for the predicate iteratively at each step

IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Running time of the Algorithm

0. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
1. Start with a segmentation S^0, where each vertex v_i is in its own component.
2. Repeat step 3 for $q = 1, \ldots, m$.
3. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_i^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
4. Return $S = S^m$.

Step 0 can be done in O(m) time

Step 1-3 Can be done in O(ma(m)) time where a is the inverse ackerman's functions because checking for a predicate is completed in constant time and finding the root and merging them can be accomplished in a(m) time

$$\mathrm{F}(4, n) = 2[4]n = 2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$$

a(m) <= m for practical purposes

Existence of a Segmentation

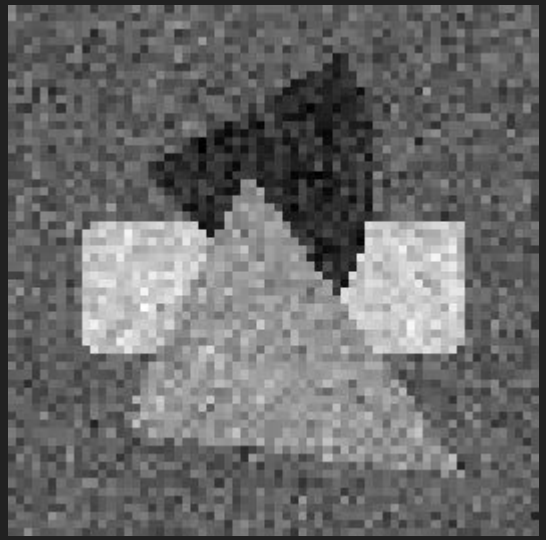
Def 1) A segmentation is too fine if no evidence of boundary between two components C_i, $C_j \in S$

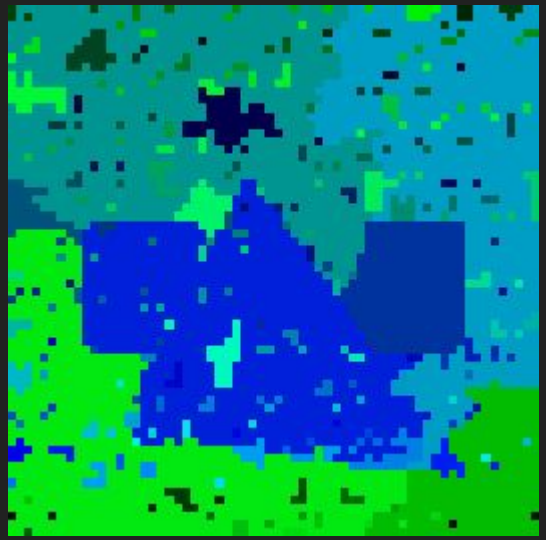
Def 2) A segmentation is too coarse if there exists a proper refinement of S that is not too fine

There always exist a segmentation that neither too coarse nor too fine

The proof is quite trivial to see

Implementations





Implementations