Felzenszwalb, Pedro F., and Daniel P. Huttenlocher. "Efficient graph-based image segmentation." *International Journal of Computer Vision* 59.2 (2004): 167-181.

Rough Outline

1. Motivation / Problem Definition

2. Contributions of the paper
   - ‘Evidence of boundary’ predicate
   - Definitions of “Too fine” and “Too Coarse” partitions
   - Greedy algorithm
   - Proof of why algo gives a ‘not too fine’ and ‘not too coarse’ segmentation

3. Implementation & Running time
4. Grid graphs (images) and Nearest neighbor graphs

Introduction to the Problem

Definition of a Segmentation

Partitioning an image into sets of pixels

Image I as a set of pixels $I = \{p_i\}$

Segmentation S is a partition of I $S = \{C_i\}$

such that, $C_i \bigcap C_j = \phi, i, j = 1 \ldots |S|, i \neq j$

- And $\bigcup_{i=1}^{|S|} C_i = I$

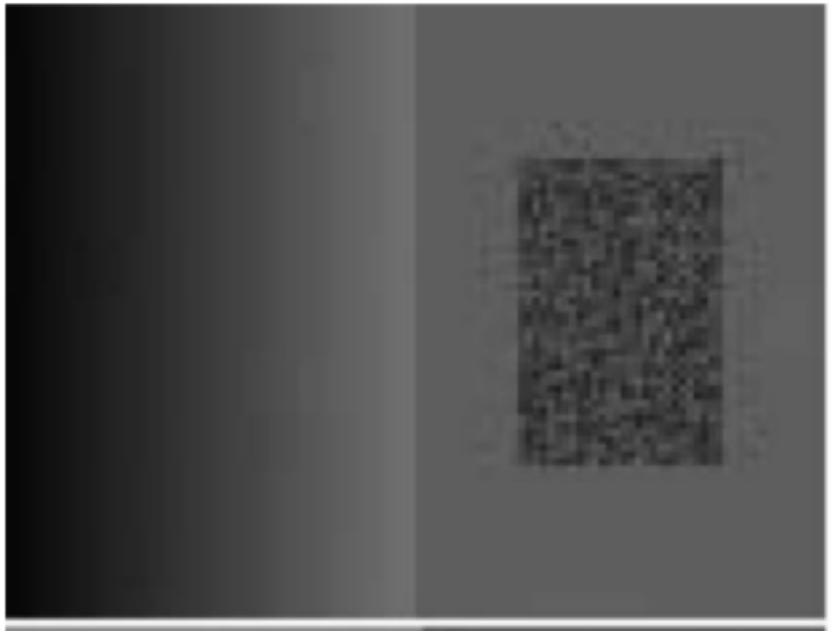
Image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share **certain perceptually similar** characteristics.

Motivation

Proposed algorithm must capture perceptual important image properties

3 perceptual regions

- Intensity ramp (left part)
- Constant intensity
- High variability region



Note : Varying intensity should not alone be judged as evidence for multiple regions

Note : 3 different regions cannot be obtained using purely local decision criteria

Intuitively, intensity difference across the boundary of the regions are perceptually important if they are large relative to intensity differences inside the region.

Evidence of boundary :

- Intensity difference across the boundary (External Difference)
- Intensity difference between pixel within a region (Internal Difference)

Region Comparison Predicate

Generic Graph-based Segmentation

G : Undirected graph, with $G = (V, E)$
V : set of elements to be segmented $v_i \in V$
E : set of edges. Measure of dis-similarity between neighboring elements.

$(v_i, v_j) \in E \qquad w((v_i, vj))$

Segmentation : A sub-graph induced by subset of edges

Segmentation $S = \{C_i\}$, C_i is a connected sub-graph of G' = (V,E') **induced by subset of** edges of G $E' \subseteq E$. *Give an example*

Problem : To find a subset which satisfies some perceptually important properties

Related Definitions

Definitions:

1. Internal difference of a component :
   - Largest weight of the minimum spanning tree of the component

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

   - Intuition, C remains connected when edges of weight at most Int(C) are considered
   - This definition provides an efficient way to construct components and is helpful in the proof (coming later)

2. External difference between 2 components
   - Minimum weight edge between them.

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

   - If there is no edge between a pair of component ext-diff = infinity

Pairwise Region Comparison Predicate

Predicate D : Evidence for a boundary between a pair of components C1, C2

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \quad (3)$$

External Difference

Internal Difference

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)). \quad (4)$$

Note : \tau controls the degree to which the difference between the 2 must be greater than Int diff. $\tau = \frac{k}{|C|}$

D is true if external difference between 2 components is relatively larger than min-components.

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \tag{3}$$

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)). \tag{4}$$

If there is no edge between a pair of component ext-diff = infinity

More explanation about \Tau

D is true if ext-diff is larger than int-diff by atleast \tau.

Case 1 : $Int(C_1) + \tau(C_1) < Int(C_2) + \tau(C_2)$

Case 2 : $Int(C_1) + \tau(C_1) \geq Int(C_2) + \tau(C_2)$

For smaller components, Int(C1) is not a good estimate of the internal characteristics of a component.

Thus, we require a stronger evidence for a boundary for smaller components. So \tau is inversely proportional to size of components.

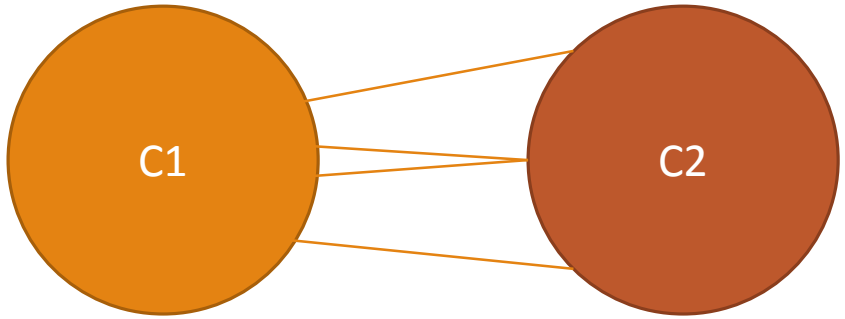
Definition of ‘too fine’ & ‘too coarse’

Ultimate goal is to find a segmentation that is not ‘too fine’ and not ‘too coarse’

Segmentation is ‘too fine’

A segmentation is *too fine* if there is some pair in regions for which D is false

$$\exists\, C_i, C_j \in S, i \neq j, D(C_i, C_j) = false$$

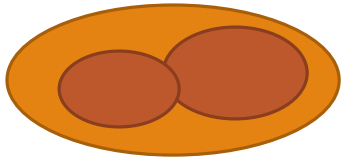


Not too fine ➔D is true for every pair of components

$$\forall C_i, C_j \in S, D(C_i, C_j) = true$$

Segmentation is ‘too coarse’

1. S and T be 2 segmentations of the same base set.
2. T is a refinement of S when each component of T is fully contained in some component of S
3. When T is refinement of S, it is called, “T is finer than S” or “S is coarser than T”



A segmentation S is too coarse when there exists a refinement that is not too fine

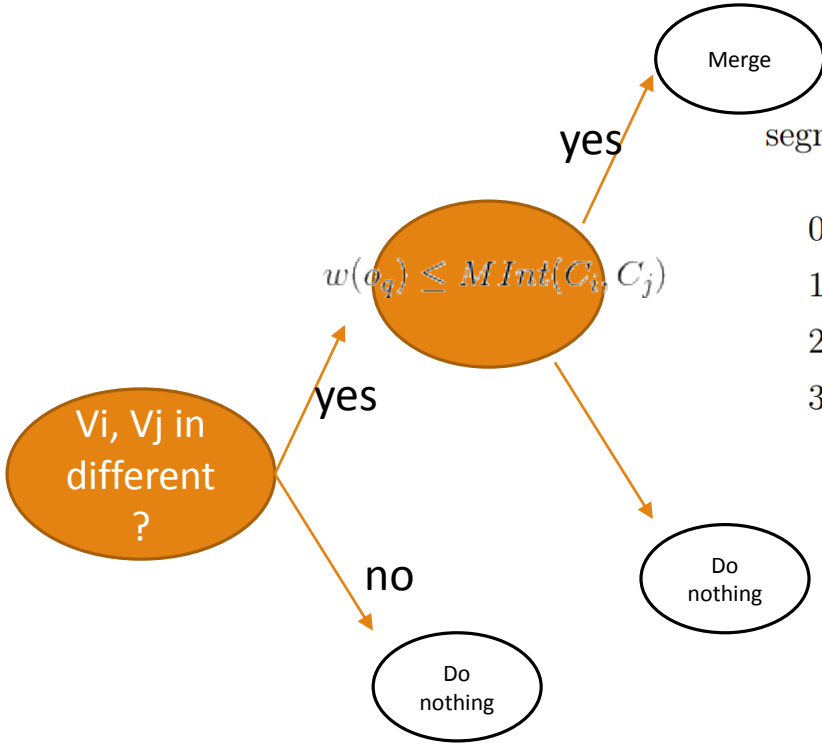
Ie. There is an evidence for boundary (D=true) within a component, than the segmentation has too few regions.

Proof of :
For any G, there exists some segmentation, S, that is neither too fine, nor too coarse

1. Consider a segmentation, S, where all elements are in only 1 component. Thus S is not too fine
2. If S is also not too coarse then we are done
3. If S is too coarse, there exist a refinement of S, that is not too fine.
4. Repeat this argument with the refinements of S.

Algorithm

Algorithm – Pseudo Code



The input is a graph $G = (V, E)$, with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, \ldots, C_r)$.

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 $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ 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 $S = S^m$.

To Prove

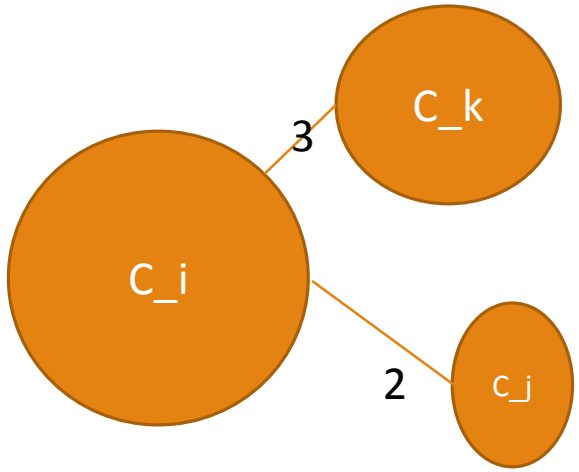
1. Algorithm gives a segmentation that is
   - neither too fine
   - nor too coarse
2. Any possible non-decreasing weight ordering will produce same segmentation

“if 2 components are not merged, then one of them will be in final segmentation”

There are 2 cases in which merge do not happen --

Case 1: $w(o_q) > Int(C_i^{q-1}) + \tau(C_i^{q-1})$

Since edges are considered in ascending order w(o_k) > w(o_q) (where k>q). Thus no additional merge shall happen to this components.



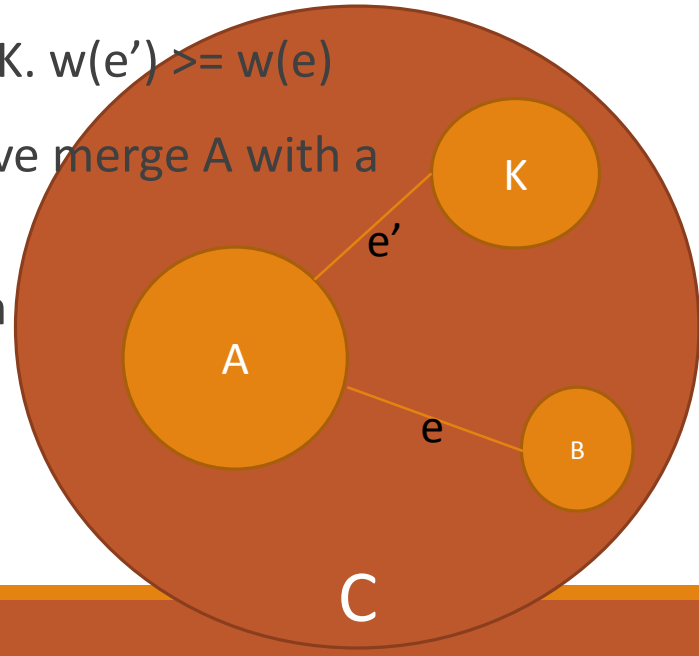
NOTE: Edge causing the merge of 2 components is exactly the min weight edge between the component

Segmentation S by the algorithm is not ‘too fine’

1. From previous property, if Ci and Cj are different component, there is an edge (which was considered) and did not cause a merge
2. Since it didn’t cause a merge w(o_q) > Mint(Ci, Cj). Note that o_q is the minimum edge between the components, ie. Ext-diff
3. This implies D is true.
4. If D is true for every pair of components, implies that the segmentation is not too fine.

Segmentation S by the algorithm is not ‘too coarse’

1. Let S (segmentation produced by algo) be ‘too coarse’.
2. Thus, for a component C, there exists a refinement which is not ‘too fine’. A---(e)---B
3. W(e) > Mint(A,B)
4. Without loss of generality, assume, w(e) > Int(A)+\tau(A) ➔ w(e) > max(MST(A))
5. Let K be some other sub-component of C and e’ be the edge from A to K. w(e’) >= w(e)
6. The algorithm must have formed A first before forming C. And must have merge A with a sub-component of C.
7. This is not possible since, w(e’) > Int(A) + \tau(A). This is a contradiction



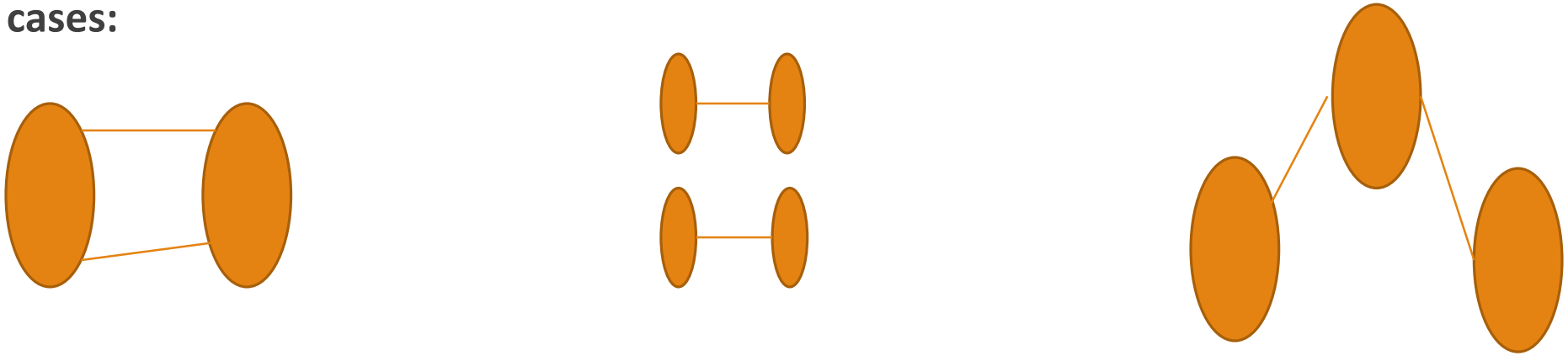
Segmentation produced does not depend on the non-decreasing weight order of the edges

Swapping the order of 2 adjacent edges of ‘same weight in non-decreasing order’
does not change the result of segmentation

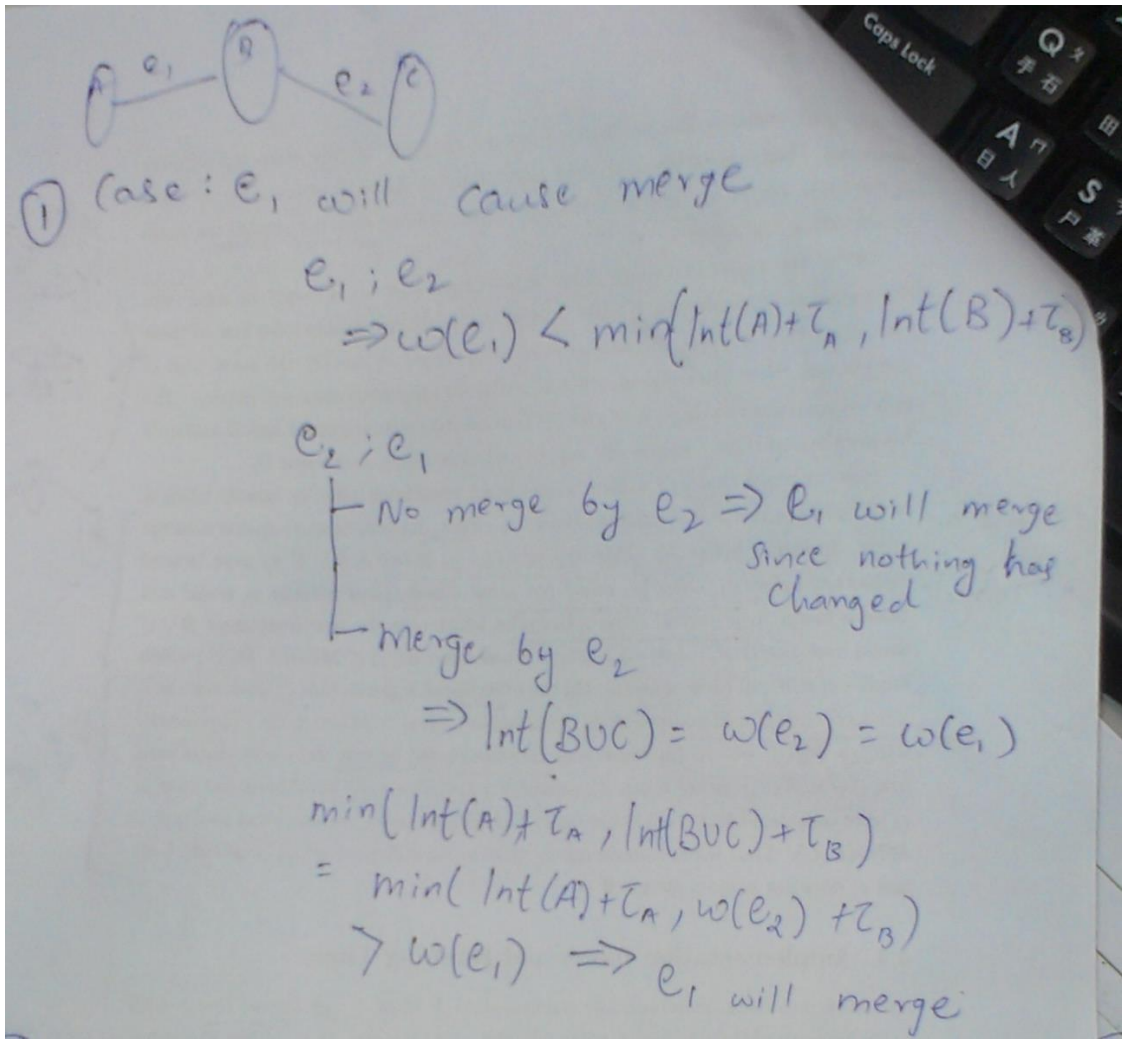
E1, e2 be 2 edges of same weight in ‘non-decreasing order’
O1, o2, o3 …. e1,e2, … om
O1, o2, o3 …. e2,e1 … om

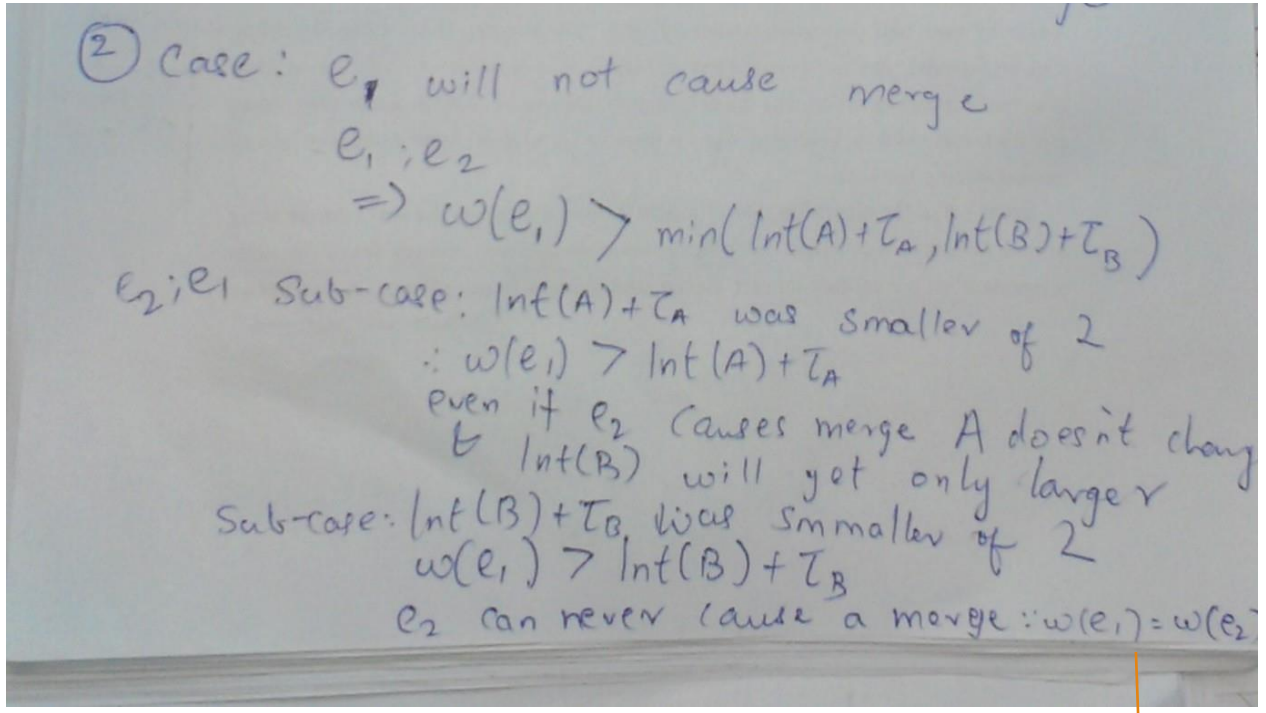
Produce the same segmentation

3 cases:



To prove: if e1 has to cause a merge it will cause the merge, no matter the ordering is e1,e2 or e2,e1
To prove: if e1 has to not cause a merge, it will not cause a merge, no matter the ordering is e1,e2 or e2,e1





=Int(B)

Implementation Issues

The segmentation can be represented in computer memory using ‘dis-joint set forest’

Analysis of running time of algo

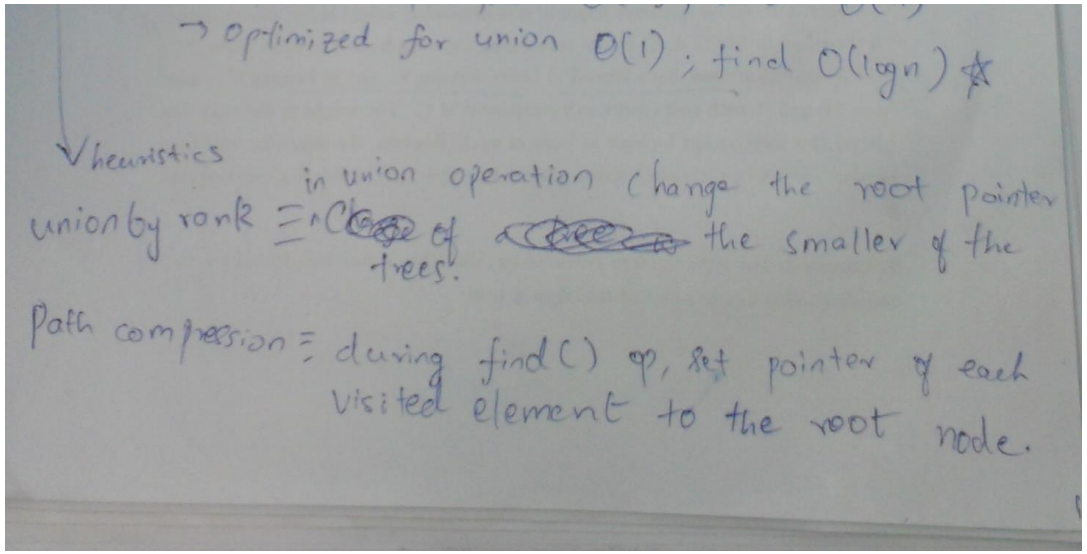
Optimized with heuristics of ‘Union by rank’ and ‘path compression’

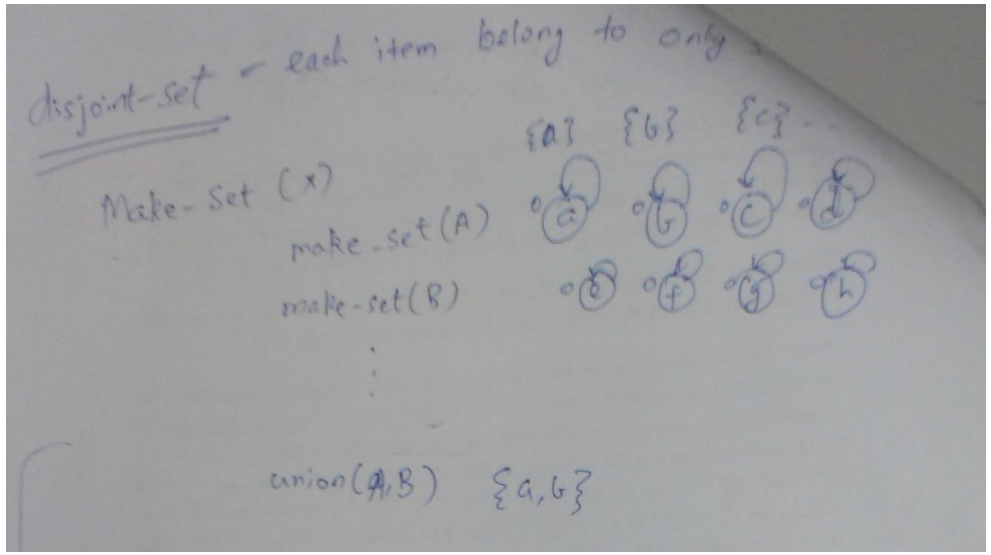
Dis-joint set

"A data structure that keeps track of a set of elements partitioned into non-overlapping sets"

Operations :

- Makeset(v1, v2, …. Vn) → init
- Find()
- Union()





Pseudo Code

1. {PI} = {o1, o2, … on} : Sort edges
2. M = Make-set(v1, v2, … vm) : set each vertex in different set
3. For each {PI}
   i. Vx = oi.vx ; Vy = oi.vy
   ii. a= Find-set(Vx, M)
   iii. b= Find-set(Vy, M)
   iv. Compare a, b. if equal ➔ both vertices in same set, thus goto 3. else v.
   v. Int-vx = MST(a)
   vi. Int-vy = MST(b)
   vii. If w(oi) < Mint(vx, vy) then set-union(vx, vy)
4. The resultant set represents a segmentation of the vertices

Derive the running time of algo

(time to sort m edges) + (m * T{find-set()}) + (m*T{MST}) + (m * T{union-set})

= O(m logm) + O(m*logn) + O(m) + O(m)

= O(m logm) + O(m logn)

= O(m log mn)

= O(n log n) → almost linear time in # of vertices

Results for Grid Graph

Graph constructed based on spatial neighbors

W(vi,vj) = | I(pi) – I(pj) |

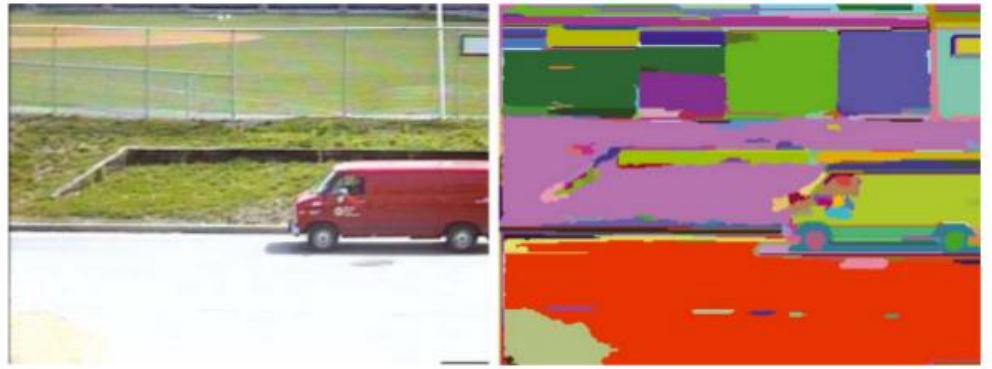


Figure 2. A street scene (320 × 240 color image), and the segmentation results produced by our algorithm ($\sigma = 0.8, k = 300$).

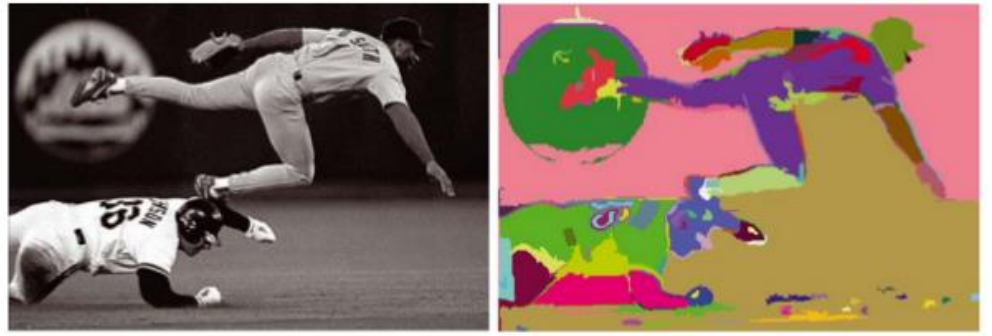
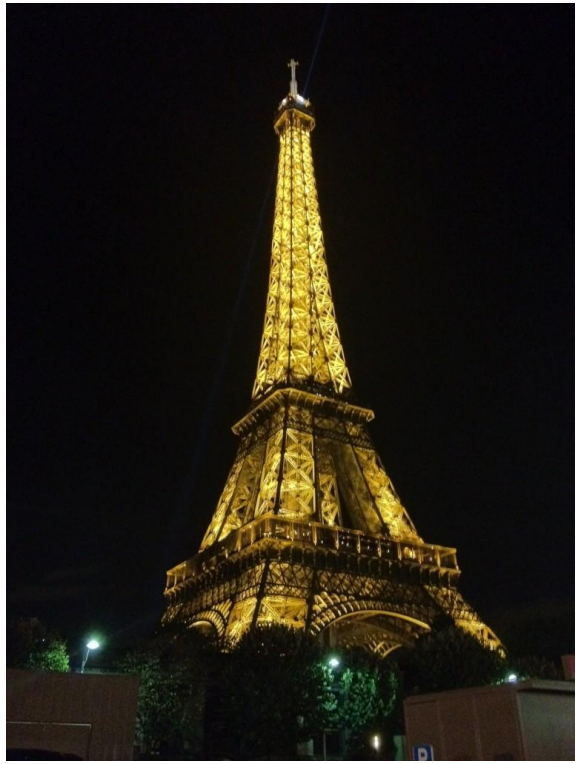
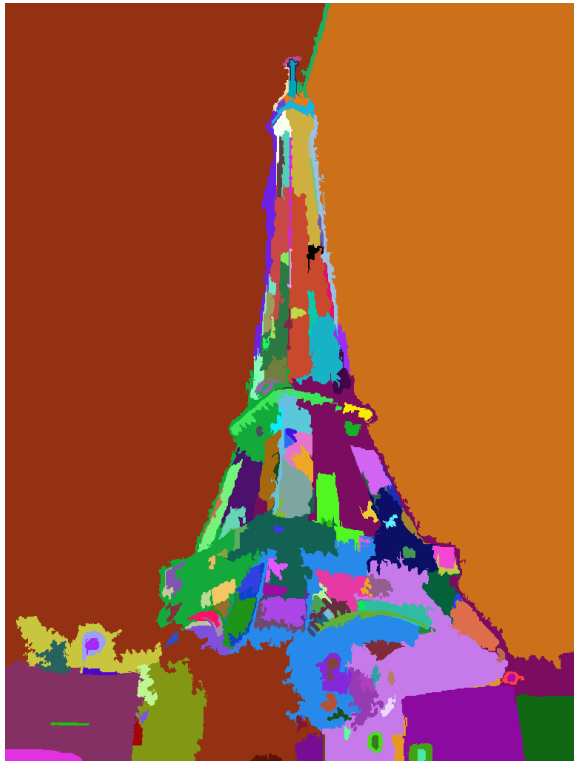


Figure 3. A baseball scene (432 × 294 grey image), and the segmentation results produced by our algorithm ($\sigma = 0.8, k = 300$).

Figure 4. An indoor scene (image 320 × 240, color), and the segmentation results produced by our algorithm ($\sigma = 0.8, k = 300$).

Failure case – Grid Graphs





Results for Nearest Neighbor Graph

Mapping each pixel to a point in some feature space

Connect edges between nearby vertices in the feature space

Alternately, connect all vertices which are at distance less than d (in the feature space)

Experiments : (x,y,r,g,b) as feature vector. L2 norm as distance measure.
Motivation : Eiffel tower – night image

Figure 8. Segmentation using the nearest neighbor graph can capture spatially non-local regions ($\sigma = 0.8$, $k = 300$).

Results generated by me

Parameters:
Sigma=0.5 ; k=500 ; compo_size_thresh =50

Appendix – I (Proof of Kruskal's Algo)

Lemma : If an edge (u,v) is min wt edge across a partition of vertices, then (u,v) is in MST

Proof of Kruskal's
Part - 1 : Prove that algo produces a tree
Since it collects the edges whose vertices are not in same set ➔ forms a forest/tree

Part – 2 : Tree produced is minimum
Algo selects the first weight between 2 different trees ➔ MST, since it considers edges in ascending order and the first edge encountered between trees is the minimum edge

Appendix – II (# of comparisons for a find-set() is <= log(n)

1. (10pts, 1 page)

   Prove that every node has rank at most $\lfloor (lg\ n) \rfloor$ in union-Find algorithm with two heuristics (1) union by rank and (2) path compression.

   Sol:
   By Induction:
   Claim:
   A node with rank r, then it is the root of a subtree of size at least 2^r.
   Base case:
   A node of rank 0 is the root of a subtree that contains at least itself (and so is of size = at least 1).
   Inductive case:
   A node X can have rank $(r + 1)$ only if,at some previous stage, it had a rank r and it was the root of a tree that was joined with another tree whose root had rank r. Then X became the root of the union of the two trees. Each tree, by inductive hypothesis is of size at least 2^r, and so now X is the root of a tree of size at least $2^r + 2^r = 2^{(r+1)}$.

   Now the number of nodes in the forest is n and we have at least 2^r nodes in every tree with rank r.So,

   $n \geq 2^r \Rightarrow r \leq \lfloor (lg\ n) \rfloor$