Efficient Graph-Based Image Segmentation

Felzenszwalb and Huttenlocher

Overview

- Goals:
  - Capture Perceptually important Groupings
  - Be highly efficient

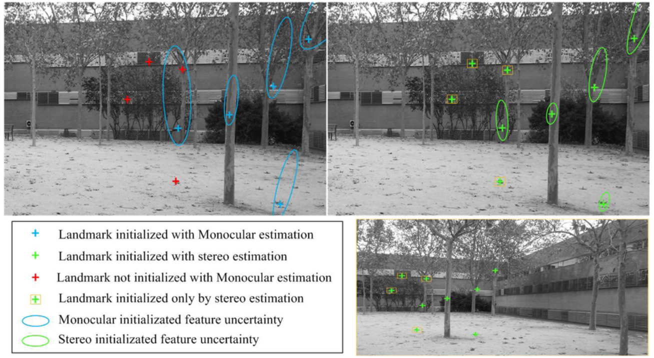
- Contributions:
  - 2 Graph Based representation of an image.
  - Greedy Algorithm (linear in number of edges in graph).
  - New Definitions to evaluate quality of segmentation.

Problems that are addressed

1. How to segment an image into regions?
2. How to define a predicate that determines a good segmentation?
3. How to create an efficient algorithm based on the predicate?
4. How do you address semantic areas with high variability in intensity?
5. How do you capture non-local properties in an image?

Applications for Segmentation

Stereo and motion estimation.

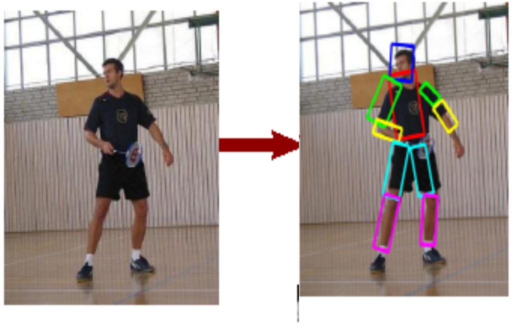


Improving recognition.

Results:

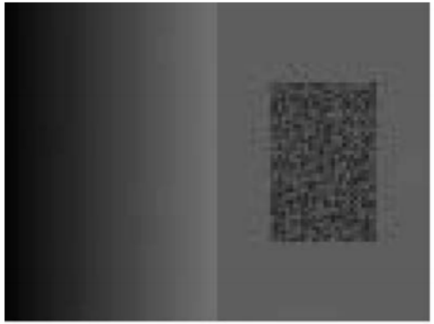
- **Doge**: 9000%
- Dog: 97.34 %
- Tibetian mastiff: 81.06%

Improving Image Matching by parts.



Main Motivation

Previous methods did not take into account that an object might have invariance in intensity and would incorrectly segment that area.



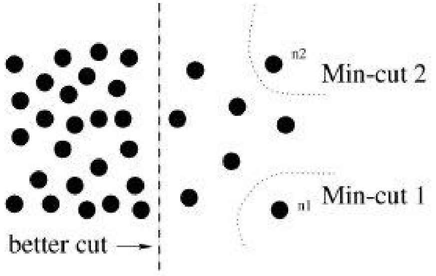
Original Image

Incorrect Segmentation

Correct Segmentation

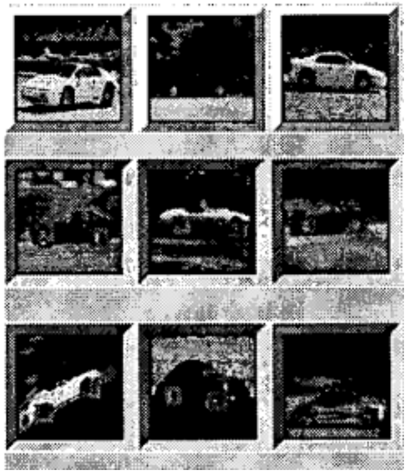
Related Works

Normalized Cuts: **Shi and Malik 1997**



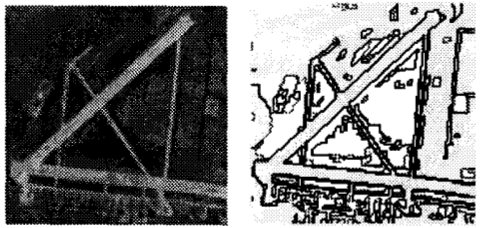
Too Slow

Ratan et. al 1997



Doesn't capture non-local properties

Minimum Cuts: **Wu and Leahly (1993)**

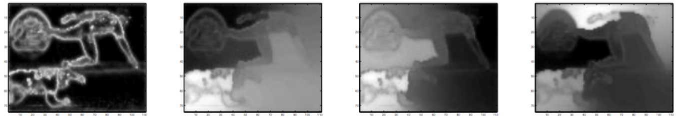


Minimizes similarity between pixels that are being split - but favors small segmentations and doesn't capture global features.

Related Works

Weiss (1999)

Eigenvector approximations to standard partitioning of graphs



Too Slow

Cooper 1998 & Pavlidas 1977

If uniformity predicate U(A) is true for a region A, then U(B) is also true for region $B \subset A$

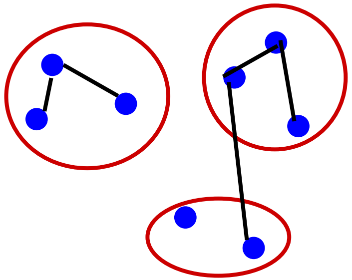
Doesn't work when uniform gradients between segments is less than inside segments

Zahn 1971

Constructs a minimum spanning tree and breaks edges with large weights.

Doesn't manage to capture areas with high variability

Problem Formulation



Graph G = (V, E)

V is set of nodes (i.e. pixels)

E is a set of undirected edges between pairs of pixels

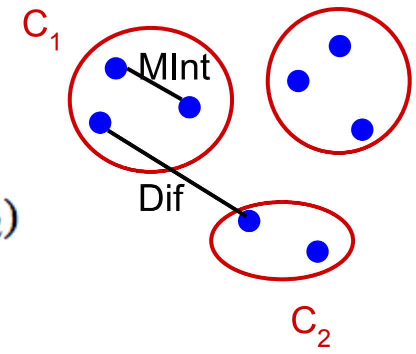
$w(v_i, v_j)$ is the weight of the edge between nodes v_i and v_j.

S is a segmentation of a graph G such that G’ = (V, E’)

where $E' \subset E$.

S divides G into G’ such that it contains distinct components (or regions) C.

Predicate for Segmentation



Predicate D determines whether there is a boundary for segmentation.

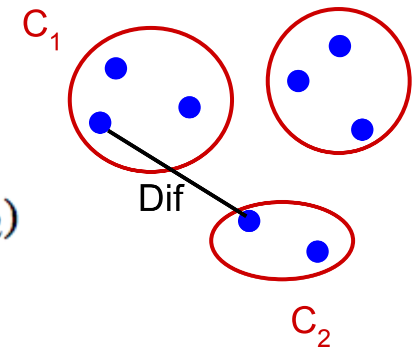
$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

Where

$Dif(C_1, C_2)$ is the difference between two components.

$MInt(C_1, C_2)$ is the internal different in the components C_1 and C_2

Predicate for Segmentation



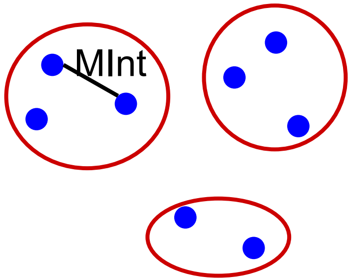
Predicate D determines whether there is a boundary for segmentation.

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

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

The different between two components is the minimum weight edge that connects a node v_i in component C_1 to node v_j in C_2

Predicate for Segmentation



Predicate D determines whether there is a boundary for segmentation.

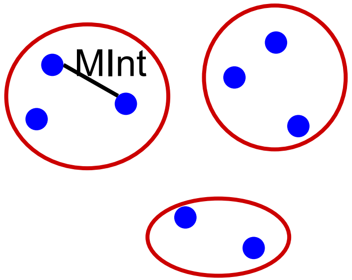
$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } \mathit{Dif}(C_1, C_2) > \mathit{MInt}(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases}$$

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

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

Int(C) is to the maximum weight edge that connects two nodes in the same component.

Predicate for Segmentation



Predicate D determines whether there is a boundary for segmentation.

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

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

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

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

where $\tau(C) = k/|C|$

Predicate for Segmentation

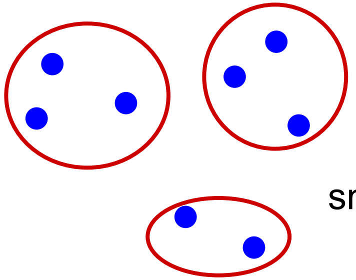
$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)).$$

where $\tau(C) = k/|C|$

T(C) sets the threshold by which the components need to be different from the internal nodes in a component.

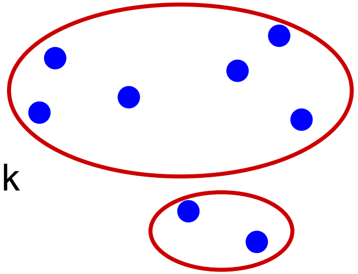
Properties of constant k:

- If k is large, it causes a preference of larger objects.
- k does not set a minimum size for components.



small k

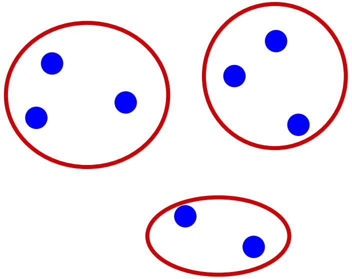
large k



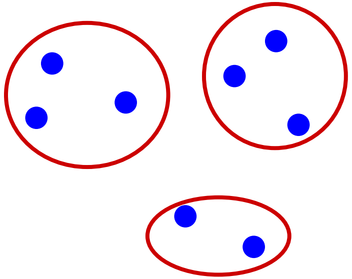
Definitions

Refinement:

For two segmentations S and T, T is a refinement of S if T can be obtained by splitting zero or more components of S.



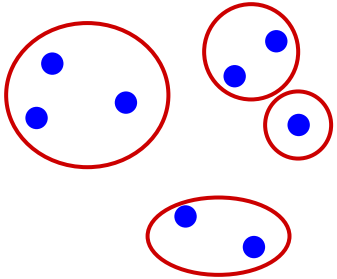
S



T: Refinement

Proper Refinement:

T is proper refinement of S if T != S.



T: Proper Refinement

Definitions

Too Fine:

S is too fine if $\exists\ C_1, C_2 \in S$ for which there is no evidence for a boundary between them.

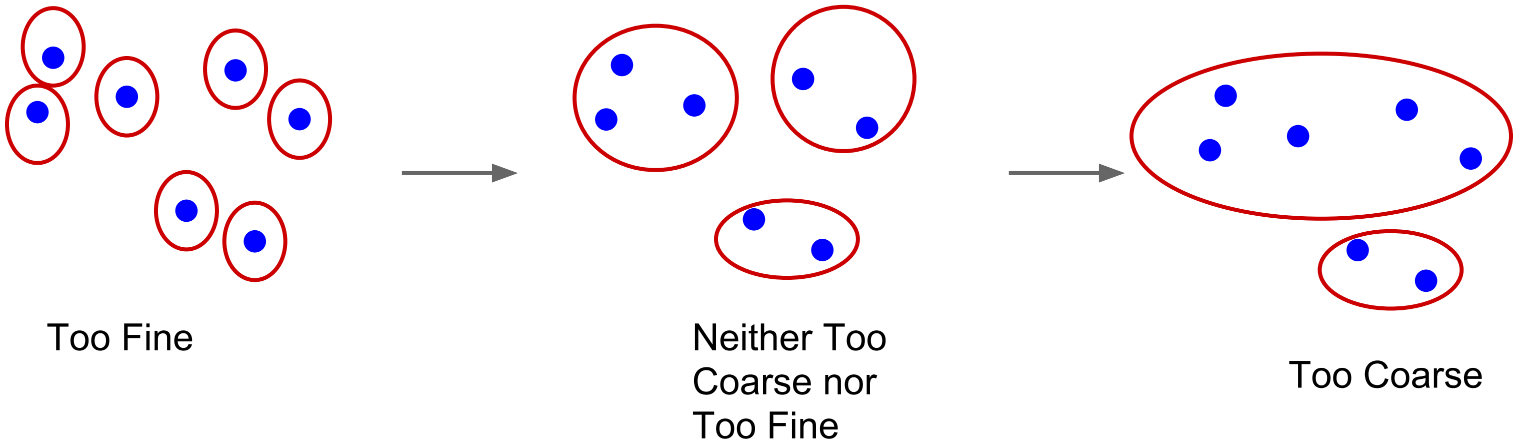
Too Coarse:

S is too coarse when there exists a Proper Refinement of S that is not Too Fine.

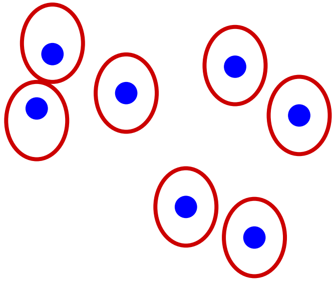
Property 1

For every graph G, there is a segmentation S that is neither too fine or too coarse.

Proof:

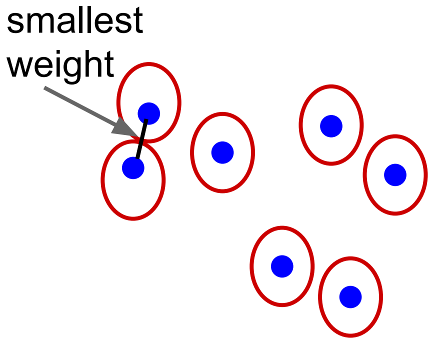


Algorithm



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

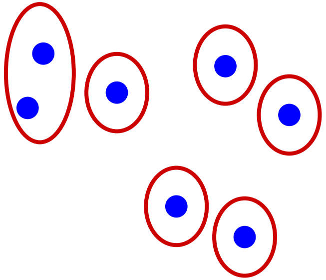
Algorithm



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 $S = S^m$.

Algorithm

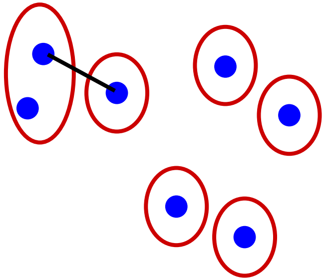
combine components



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 $S = S^m$.

Algorithm

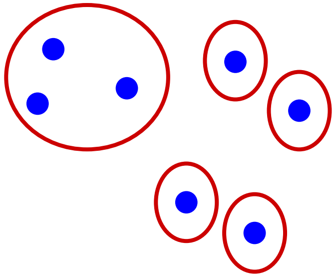
next edge



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 $S = S^m$.

Algorithm

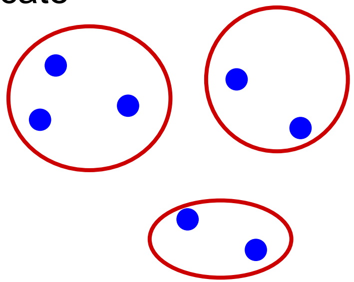
combine components



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 $S = S^m$.

Algorithm

no more edges that satisfy the predicate



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 $S = S^m$.

Lemma 1. *In Step 3 of the algorithm, when considering edge o_q, if two distinct components are considered and not merged then one of these two components will be in the final segmentation. Let C_i^{q-1} and C_j^{q-1} denote the two components connected by edge $o_q = (v_i, v_j)$ when this edge is considered by the algorithm. Then either $C_i = C_i^{q-1}$ or $C_j = C_j^{q-1}$, where C_i is the component containing v_i and C_j is the component containing v_j in the final segmentation S.*

Some helpful formulae: (Proof on the board)

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

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

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

Theorem 1. *The segmentation S produced by Algorithm 1 is not too fine according to Definition 1, using the region comparison predicate D defined in (3).*

Some helpful formulae: (Proof on the board)

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

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

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

Theorem 2. *The segmentation S produced by Algorithm 1 is not too coarse according to Definition 2, using the region comparison predicate D defined in (3).*

Some helpful formulae: (Proof on the board)

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

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

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

Theorem 3. *The segmentation produced by Algorithm 1 does not depend on which non-decreasing weight order of the edges is used.*

Some helpful formulae: (Proof on the board)

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

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

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

Datasets Used

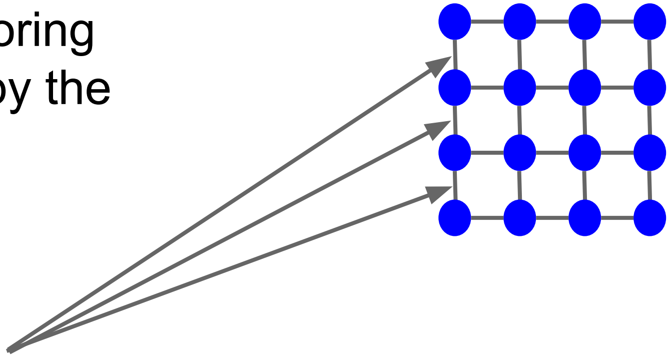
[Columbia Coil Dataset:]

- k = 150 for 138 X 138 images.
- k = 300 for 320 X 240 images.

Columbia University Image Library
(COIL-20)

Grid Graph Weights

Every pixel is connected to its 8 neighboring pixels and the weights are determined by the difference in intensities.

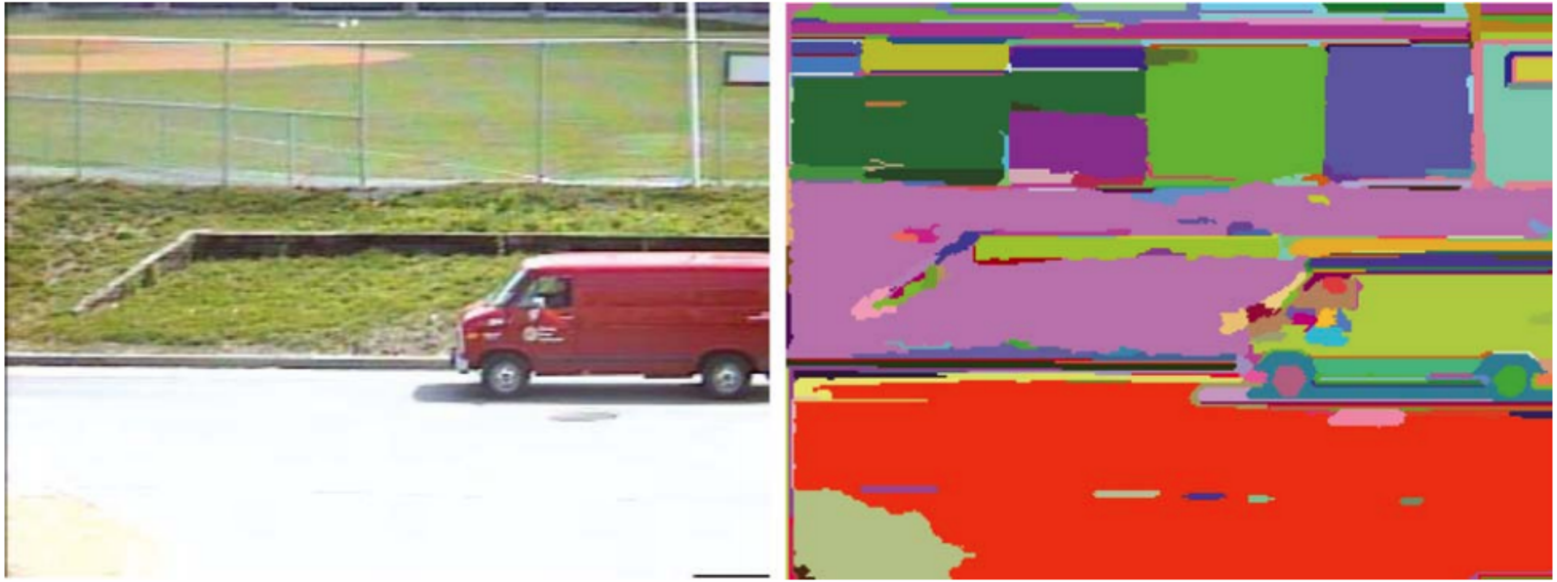


$$w(v_i, v_j) = |I(p_i) - I(p_j)|$$

For color images, they run the algorithm three times using R values, then using G values and finally B values.

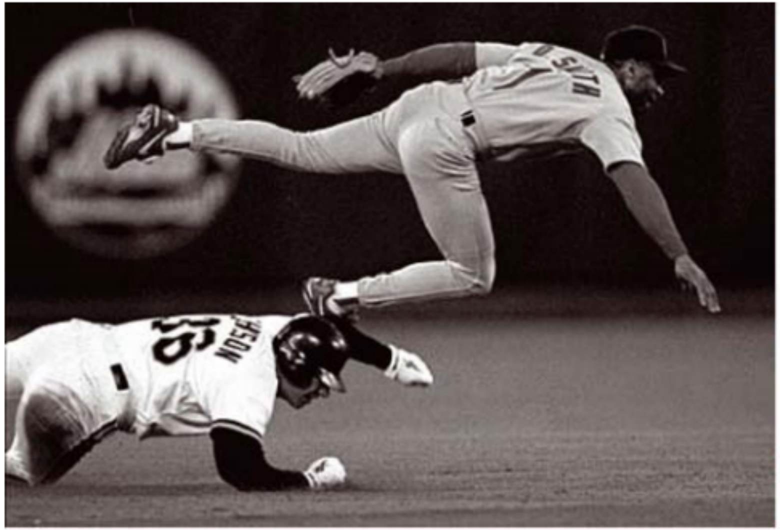
They put two pixels in the same component only if they appear in the same component in all three colors.

Grid Graph Results



- The highly-variable grass gets segmented into one segment.
- Because of image artifacts, the lower left corner of the road is incorrectly segmented.
- Specular reflections of van leads to multiple segments.

Grid Graph Results



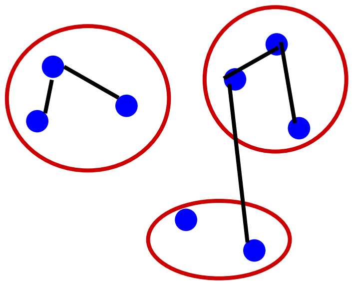
- grass and clothes with variations each have their own component.
- Due to long slow change in intensity from grass to black area, it gets mis-segmented into one component.
- Preserves small components like name tags and numbers.

Nearest Neighbor Graph Weights

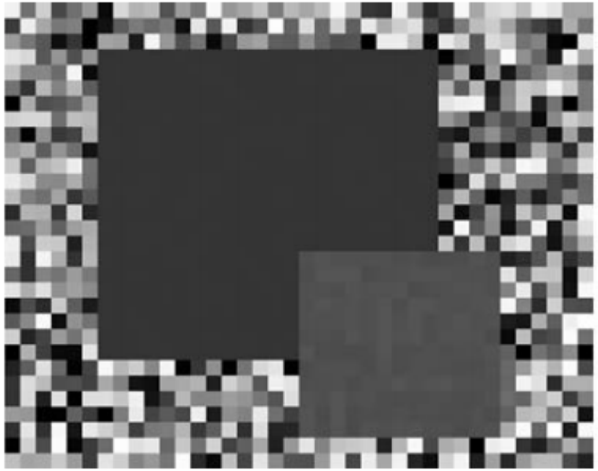
Project every pixel into feature space defined by (x, y, r, g, b).

Weights between pixels are determined using L_2 (Euclidian) distance in feature space.

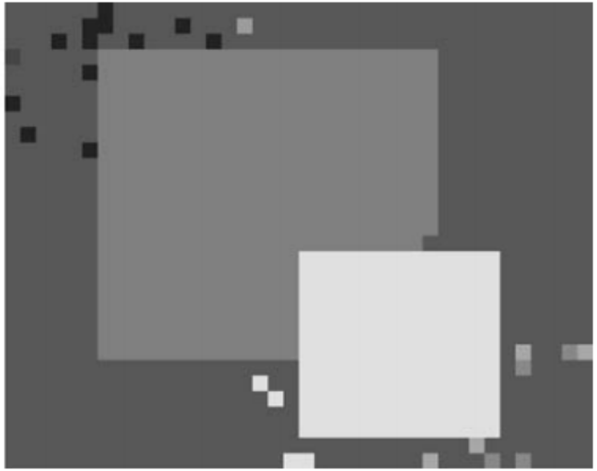
Edges are chosen for only top ten nearest neighbors in feature space to ensure run time of $O(n \log n)$ where n is number of pixels.



Nearest Neighbor Graph Results



Image



Segmentation

- Highly variable region is placed in one large segment.
- Captures global image features.

Nearest Neighbor Graph Results

Non Spatially connection regions of the image are placed in the same component. For example:

- Flowers on the picture to the right
- Tower and lights on picture to the right.

Conclusion

1. How to segment an image into regions?

   Graph G = (V, E) segmented to S using the algorithm defined earlier.

2. How to define a predicate that determines a good segmentation?

   Using the definitions for *Too Fine* and *Too Coarse.*

3. How to create an efficient algorithm based on the predicate?

   Greedy algorithm that captures global image features.

4. How do you address semantic areas with high variability in intensity?

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

5. How do you capture non-local properties in an image?

   Nearest Neighbor approach in feature space.