A Basic Approach to:

Binary Image Object Segmentation

Victor Tellez



# **Assumptions**

Below are some assumptions that will be made through out the document.

* **pic1:** is the 150 x 150 x 4 uint8 matrix of the sample image, Similar to the one above. NOTE: the Image has objects labeled in WHITE while the background is BLACK.

**Variables Used:**

* **B:** This is a 150 x 150 matrix representing our input image, pic1, each cell can range in value from 0 to 255 unsigned
* **f:** An array that is our representation of a Union-Find data structure. This allows us to identify how many objects there are in our image
* **numbersFound:** The number representation that tracks how many segments the algorithm discovers in the picture.
* **NumberOfObjects:** The number that represents how many objects there are in our image after unifying connected segments.
* **labels:** An array that holds the unique labels found in the image.

# **Step\_1: [B,f] = AssignLabelsRound1(B,f)**

This function labels all non-black pixels and maintains a Union-Find data structure in order to keep track of the ancestry of every pixel segments we identify.

**Function Variables**

* **B:**  This is the matrix representation of our Grayscale Image
* **f:** This is array will initially be empty.

The function starts by initializing the variables:

* **m:** the number of rows in the picture B
* **n:** the number of columns in the picture B
* **k:** The variable we use to assign number labels. Every time we encounter a new segment we increment k by one and label all the pixels in this segment with the new k value. We initialize k to 2, because 1 is the Null value in our array and it is how we terminate our search in the union-find. (NOTE: In MatLab arrays start at 1) So 2 is the first possible node of a unique object.

**What happens in the function:**

For every pixel in the image B; where variable “i” keeps track of the current row and, variable “j” keeps track of the current column.

If the current pixel is white, where a pixel value of 255 is used to represent white. B(i,j)==255.

Initialize variable a, b, c. We will be using this ‘window’ in order to segment our image.

a = B(i , j); b = B(i , j-1); c = B(i-1 , j);

* If a’s neighbors, c and b, are both black pixels, then we have encountered a new segment of pixels. Thus we create a new label by incrementing k by one and assigning the current pixel B(i,j) the value k
* Else if b is black but c is not, label pixel B(i,j) with the label at c.
* Else if c is black but b is not, label pixel B(i,j) with the label at b.
* Else if b and c have equal labels that are not black pixel values. Then label pixel B(i,j) with the label at b.
* Else if b and c have non-black pixel values but do not have the same labels. Then evaluate b and c to see if they share an ancestor.
  + If b and c share an ancestor then label pixel B(i,j) with the label at b.
  + If b and c do not share an ancestor, then Unite the trees of b and c so that they are part of the same tree structure and share an ancestor. After this label pixel B(i,j) with the label at b.

# **Step\_2: [B] = AssignLabelsRound2(B,f)**

This function unites adjacent segments in the picture and maintains their relationship in the Union-Find data structure. Making it so that any label in the array ‘f’ holds the value of its oldest ancestor.

**Function Variables**

* **B:** The image matrix now has been segmented into pixel regions which are defined by the label that each pixel holds.
* **f:** This array is now holding information about what segmented regions are adjacent to each other and therefor are actually the same region.
* **k:** This variable is used to momentarily keep track of the current pixel label

The function starts by initializing the variables:

* **m:** the number of rows in the picture B
* **n:** the number of columns in the picture B

**What happens in the function:**

For every pixel in matrix B:

* If B(i,j) is not a black pixel, then let k = B(i,j). Starting at f(k) backtrack through the Union-Find structure until we reach the root node f(k)=0. Every time we move up to an ancestor we relabel k to be f(k) until eventually we end up at the root node.
* This has the effect of having adjacent segments pointing to each other as ancestor. Since the relationship of adjacency is undirected we are able to create directed relationships of ancestry without having any repercussions. Meaning that it does not matter that segment X is the parent of segment Y; what does matter is that the relationship is documented since they are adjacent to each other.
* We begin by letting k = B(i,j), this function then checks to see if k has an ancestor by looking at the Union-Find structure. A segment has no ancestors when f(k)= 0. When we determine what segment will be the common ancestor to all other adjacent segments we set pixel B(i,j)= k, where k is the common ancestor segment. So if the original label stored in B(i,j) we change it to be the label of its oldest ancestor.
* Then we run through the ‘f’ Union-Find structure and replace any decedent node value with the value of its oldest ancestor. Meaning that the data structure goes from storing different labels and their ancestors to only storing the value of the oldest ancestor at any position where a descendent would have been. For example, if we had two objects Object A with segments 2,3,4 where segment 2 is the oldest, and Object B with segments 5,6,7,8 with 6 being the oldest. Then the Union-Find would go from looking like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array f Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| f(i) value | 0 | 1 | 4 | 2 | 6 | 1 | 6 | 7 |

* To looking like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Array f Position | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| f(i) value | 0 | 1 | 2 | 2 | 6 | 1 | 6 | 6 |

# **Step\_3: [numbersFound, NumberOfObjects] = ObjectCount(B)**

This function identifies the number of unique objects in the image B.

**What happens in the function:**

For every pixel in the B matrix that is not equal to 0, set the value in numbersFound(B(i,j)) array equal to 1.

Add every value found in numbersFound array to obtain the number of unique objects in the image.

# **Step\_4: [labels] = FindLabelsUsedInPic(**

# **NumberOfObjects, numbersFound )**

This function identifies the labels that represent each object in the image. The labels array will have the numeric labels that each image can be identified by.

**What happens in the function:**

For every pixel in the image B if we encounter a unique value that is not in our labels array and it is not the value zero, then we add this value to our labels array.