Minimum Spanning Tree-based Image Segmentation and Its Application in Cutout Filter

Yonas Adiel Wiguna - 13516030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
ya@students.itb.ac.id

***Abstract*—Images are actually matrices of single color combined from red, green, and blue value called pixel. To get a better grasp about information provided by the image automatically, sometimes we need a step called image segmentation. This step divides image into many segment based on color, to be processed into much more advance and applicative process such as face recognition, optical character recognition, etc. This paper discuss one of techniques used to segment an image: Minimum Spanning Tree (MST) based clustering using Kruskal's Algorithm and its application in order to apply cutout filter in the image. Cutout filter itself is a filter commonly used in graphic design to make an image looks like it's made of several layers of colored paper.**

***Keywords*—cutout filter, image processing, image segmentation, minimum spanning tree.**

I. Introduction

Image segmentation is one of interesting topic nowadays, remembering digital images with colors are available to many people. Image with colors, compared to grayscale image, can provide more information. This way, a computer can process the image and make an educational guess about the image. Whether it is fake or not, contains a face or not, reconstruct it as a text, and many more. However, an image is seen by computer as bit – sequence of 0's and 1's that defined the image itself. Eight bit define a byte; a byte define a color; red, green, andblue color define a pixel; matrix of pixel define an image. We have to make computer learn an image through some sort of algorithm. In order to do so, we have to cluster the image into some segment and simplify the image.

A classical technique for clustering is k-means algorithm [1]. A different approach is using MST, which is O(E log V) in general. The pixels are considered as nodes, and minimum spanning tree is constructed by Kruskal Algorithm [2].

This way, many advance process can be done to the simplified image. There are numerous application, such as face detection, optical character recognition (OCR), biometrics, etc. One of the simplest application is cutout filter in graphic design. This filter minimzed usage of colors in a picture. Gradient and similiar color are grouped into one same color. This is commonly used in some design, even one of popular image editing program includes this filter.

II. Graph, Trees, and Minimum Spanning tree

A. Graph

A *graph* $G = (V, E)$ consists of V, a nonemptu set of *vertices* (or *nodes*) and E, a set of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints [3]. Two vertices u and v are *adjacent* to each other if and only if there is at least one edge e that connect those two vertices. Then, e can be called *incident* with vertices u and v and representated as (u,v).

There are some special graphs. A graph where all of its vertices are adjacent to all other vertices are called *complete* graph. Tree is graph with special properties that will be explained in next section.

Graph can be categorized by its properties. Weighted graph is a graph where every edges has a value assigned. In the other hand, unweighted graph is a graph without values on their edges. One can see a unweighted graph as weighted graph with all of its edges have same value. Then, a directed graph is a graph where every edges has direction. One of the endpoint is the source, and the other endpoint is the destination. Then, an edge e with source u and destination v is connecting u to v. Then, if u is connected to v, v is not necessarily connected to u. Undirected graph is the opposite; no direction in edge. Thus, if u is connected to v, v must be also connected to u.

We can present graph with some representation. There are some representation, such as adjacency list, adjacency matrix, and edge list. There are some other representation that are not mentioned, because it is not really relevant to this paper.

Adjacency list is assigned every vertices with a list of its neighbors. Then, a vertex v is connected to vertex u if and only if u is listed on v's adjacency list. If the graph is weighted, the adjacency list will be populated with tuple (u, w) where w is the weight. In programming, this representation is useful when we want to traverse the graph, such as flood-fill or BFS algorithm.

Adjacency matrix is a matrix that define the graph. Dimension of the matrix is the number of vertices for both matrix row and columns. Every element of the matrix is either 1 (one) or 0 (zero). Then, an edge (u,v) exist in the graph if and only if element $A_{u,v}$ is 1. Otherwise, element $A_{u,v}$ is 0. In weighted graph, the element $A_{u,v}$ can be replaced as the weight. In programming, this representation is useful when we need to

check whether an edge is exist or not, or whether a vertex is directly connected to other vertex or not.

Edge list is a list that contains all of the edges in a graph. A vertex u is connected to vertex v if and only if there is a tuple (u,v) in the list. If the graph is weighted, the tuple will have 3 element, with the third element is w, the weight of the edge. In programming, this representation is useful when we need to iterate all of the edges in a graph.

A graph $G' = (V', E')$ is a subgraph of graph $G = (V, E)$ if and only if V' is subset of V, and E' is subset of E [3]. A subgraph can be generated from w graph with removing some of its edges, or removing some of its vertices with their incedent edges.

A path is a sequence of vertices with every two adjacent vertices in this sequence have edge between them. A path is said simple path if the sequence can be traversed without using any edge more than once. A circuit is a path where its beginning and the end is the same vertex. Again, simple circuit is a circuit when the sequence can be traversed without using any edge more than once.

B. Trees

A tree is a connected undirected graph with no simple circuit [3]. The term connected graph means for every vertices u and v, there is some path between them. With a little observation, we can found that for every two vertices, there are only one unique path. A tree with V vertices has $V - 1$ edges. This theorem can be proved by induction proof.

A graph consist of multiple trees are called a forest. Formally, a forest is an undirected graph without simple circuit. A forest can be made with removing one or more edges from a tree.

Some of the tree have root. When a tree have a root, every vertex have a parent, except the root itself. Every edge are directed away from the root [3]. This way, all vertices who have parent of a vertex are called as children. Sibling are term for vertices with same direct parent. If there is a path away from the root from vertex u to vertex v, v is called as descendant of u. The reversal, u is ancestor of v. Of course, all vertex are the descendant of the root except the root itself. Vertices who don't have child are called as leaves.

Some rooted trees are called n-ary tree, if every vertices has either no child or n child except one vertices. Full n-ary tree with i internal vertices contains $m = ni + 1$ vertices.

A spanning tree is made from a graph with removing its edge without disconnecting its vertices. Formally, a spanning tree is a tree containing every vertex of its graph.

A spanning tree can be made with DFS or BFS algorithm. These two algorithm are often used to traversing a graph. BFS focused on visit nearest vertices from source first, before advancing further. Meanwhile, DFS focused on traversing until the current vertex doesn't have any other neighbor that have been visited. BFS can be used on general graph, while DFS are limited to tree or other special case of graph. Both algorithms are solution for Single Source Path (SSP) traversing.

C. Minimum Spanning Tree

A Minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges [3]. This problem is useful on network design, planning traffic/road, clustering, and more. There are two famous algorithms that are used to calculate the minimum spanning tree of a graph: Prim's and Kruskal's.

Prim's algorithm begins by choosing a starting vertex, flag it as taken, and enqueues a pair of information int a priority queue: the weight and the other endpoint for every edge incident with the starting vertex. Then, while the priority queue isn't empty, we take top element of priority queue with minimum weight and add the edge into MST. Flag the other endpoint with visited, and repeat this step until all of the vertces are flagged as taken or $V - 1$ edges have already taken with V is number of vertices. [4]

The pseudocode can be found below.

```
priority_queue pq sorted desc
visited ← false for every vertices
MST ← empty_list()

push (0, 0) to pq
visited[e[0].source] ← true

while pq not empty:
  t ← top element of pw
  pop top element of pq
  for every e incident with t.vertex:
    if not visited[e.destination]:
      visited[e.destination] ← true
      add e to MST
      push (e.weigth, e.destination) to pq
```

This algorithm runs in O(E log V) where E is number of edges and V is number of vertices. Adjacency list representation of graph is preferred when it comes to Prim's Algorithm.

Meanwhile, Kruskal's algorithm use Union-Find Disjoint Set (UFDS). Kruskal's Algorithm begins as follows. Pick edge with minimum weight (u,v). If u and v is not connected yet in MST, add edge (u,v) to MST. Repeat those step for every edge, until all vertices are connected in MST. Edges with minimum weight are processed first [4].

```
edge_list.sortby(weight, desc)
MST ← empty_list()
i ← 0
UFDSinit()

while (MST.length <= vertex.length-1):
  edge ← edge_list[0]
  if (not UFDSsameSet(edge.from, edge.to)):
    add edge to MST
    UFDSunionSet(edge.from, edge.to)
```

This algorithm runs in O(E), excluding the O(E log V) sorting. Actually, the UFDS is affecting the MST, but its complexity is nearly constant because of path compression. Edge list representation is preferred when we want to use Kruskal's algorithm. As for UFDS, here is the implementation in pseudocode.

```
parent = []
def UFDSinit():
```

```
  for i in range(vertex.length):
    parent[i] ← i

def UFDSlookAncestor(child):
  if (parent[child] != child):
    parent[child] = UFDSlookAncestor(parent[child])
  return parent[child]

def UFDSsameSet(from, to):
  p1 = UFDSlookAncestor(from)
  p2 = UFDSlookAncestor(to)
  return p1 == p2

def UFDSunionSet(from, to):
  p = UFDSlookAncestor(from)
  parent[p] = to
```

The UFDS algorithm includes path compression algorithm, which means path from child to its ancestor is minimized. This will reduce recursive calls. Then, every time it calls `UFDSlookAncestor`, the function usually only look at its parent without doing the recursion.

III. Image Segmentation

Digital images are actually consist of pixels, aligned in certain amount of row and columns. Every pixels has three values, representing its color. Color system that is used commonly is RGB, where combination of red, green, and blue color are used to define millions of color. For example, yellow are defined as combination of red and green in nearly same amount, with nearly zero value in blue. There are some color system such as CMYK and HSL, but RGB is the commonly used in digital pictures, including our eyes. Red, green, and blue values are ranged between 0 to 255, inclusive.

In order to analyze image using graph theory, the original image must be modeled as graph. Every pixels is seen as a vertex in the graph. Its adjacent pixels are its neighbor in the graph. This is similiar to grid system in graph, where every cell is seen as single vertex and its adjacent cells as its neighbor. We can construct an undirected graph, but we still need weight to see the relation between nodes.

There are two notions to apprasie the clustering result for the data set. The first is the smaller the intra-variance in cluster of the subset is, the more similiarity the objects hold. The second is the larger the variance between clusters of the subsets is, the less similiarity the objects hold [2].

Therefore, difference between pixels are used to give every edge its weight. For example, we have two pixels u and v denoted as $(P_{u_R}, P_{u_G}, P_{u_B})$ and $(P_{v_R}, P_{v_G}, P_{v_B})$. In order to maximize the weight between two pixels with different colors and minimize the weight between two pixels with similiar colors, so we can use this formula:

$$w(u,v) = \left(P_{u_R} - P_{v_R}\right)^2 + \left(P_{u_G} - P_{v_G}\right)^2 + \left(P_{u_B} - P_{v_B}\right)^2 \quad (1)$$

With this formula, same different pixels with same RGB values will have zero (minimum) weight edge, and two pixels with different colors (any of its red, green, or blue value) will have non-zero wight. The more different the color of two pixels, the greater its weight.

However, this rule for generating the graph is still not perfect. In practice, many images, especially photos have noises. We have to handle cases where the image have noises, then we need to change the definition of adjacency of two pixels. In our new system, two pixels are considered as neighbor to each other if their manhattan distance[1] between two pixels is less or equal to a constant. We call this constant as radius, and we can find it by trial and error process.

Here is the pseudocode for graph constructor.

```
def manhattan_distance(point a, point b):
  return abs(a.x – b.x) + abs(a.y – b.y)

def pixels_distance(pixel a, pixel b):
  return (a.r – b.r)**2 +
         (a.g – b.g)**2 +
         (a.b – b.b)**2

for every v in vertices:
  for every u in vertices:
    if (manhattan_distance(u,v) <= radius):
      create edge(u, v, pixel_distance(u,v))
```

If we done with mapping to weighted undirected graph process, we can explore the graph with graph properties. However, in programming we can't understand the pixels. For program, the image is just a bunch of RGB values. The idea is creating segmentation of the image to get a better grasp of the image. Then, we can make the algotihms we needed: OCR, face detection, etc. To make the segmentation, we need to divide the image to some smaller segment with similiar color. Then, pixels with similiar color (which is defined by weight formula) are belongs to one segment.

This idea is called clustering. There are many application of clustering algorithm, especially in machine learning and data mining. There are some algorithm used in data clustering. One of the well known algorithm is k-means clustering. This algorithm see the data as n point in d-dimensional coordinate system. The data will be clustered to k cluster. This is not suit our graph representation well. The next clustering method is MST-based clustering. It as easy as it sound: we find the MST of the graph we discussed earlier. Then, we delete some of edges which has greatest weight. We repeat this step until the edge with heaviest weight is not exceeding our threshold. When we were done, we have a forest. The threshold will be found with trial and error.

To run this MST, we will need the Kruskal's algorithm. Kruskal's algorithm is preferred for MST-based clustering because of its algorithm similiarity with clustering process. Actually, the clustering can be simplified by stopping our Kruskal's when there are no more edges with weight less than our threshold. Because of Kruskal's algorithm, we prefer to use

[1] Distance between two points strictly based on sum of horizontal distance and vertical distance.

edge list to representating the graph.

Here is the pseudo code for our modified MST Kruskal's algorithm.

```
edge_list.sortby(weight, desc)
forest ← empty_list()
i ← 0
UFDSinit()

while (forest.length <= vertex.length-1 and
      edge_list[i].weight <= threshold):
  edge ← edge_list[0]
  if (not UFDSsameSet(edge.from, edge.to)):
    forest.add(edge)
    UFDSunionSet(edge.from, edge.to)
```

The result of MST-based clustering will be a forest with every tree in forest is belong to one segment (or cluster). They have edges between them, so every edges inside one segment must have weight equal or less than threshold. It will be guaranteed that for every two pixels in one segment there is unique path separating those two pixels apart with every consecutive pixels have weigth equal or less than threshold.

The rest is find the constant radius and variable threshold. Threshold is a variable, which means every image can have its own threshold. The author have tried and found value 150 for threshold variable is good enough for most of images tested. Sometimes, it is depend on the purpose of the image segmentation too. In cutout filter, the author found value 80 is more appropriate. Value 200 is fitted more when it comes to separating image from its background. In the other hand, greater the radius, better the result. But, number of edges increases quadratically. So, for better performance, it is recommended to use 2 as radius constant.

IV. Application in Cutout Filter

In graphic design, there is a filter named cutout filter. This filter can turn a photo to a photo with minimized color.

Figure 1. Example of before and after cutout filter in Adobe Photoshop CS6.

As stated in Adobe Support[2], cutout means makes an image appear as though it were constructed from roughly cut pieces of colored paper. High-contrast images appear as if in silhouette, and colored images are built up from several layers of colored paper.

The idea behind using MST-based image segmentation for cutout image filter is exploiting the result of image segmentation and change the pixels RGB value to its set average RGB value. To do that, we need to count the average of RGB values in every set. The algorithm is trivial, but we need to pay attention to how we search the set a pixel is belong. Some programming language make user have to linear search the set every time it needs to find its parent. It is recommended to make a binary search function outside the loop, or even we can precompute it before th loop. The rest of it is trivial, we can just assign the average of every set to its vertices value.

Alternative way to color all the vertices to its average is by traversing all of sets, starting from one of the element in every sets. We can use (Breadth-First Search) BFS or (Depth-First Search) DFS algorithm with same time complexity, because all of the vertices are connected to each other. But, the author won't describe it here because the trivial algorithm is fast enough and easier to code.

BFS or DFS algorithm can be used on other application of the MST-based Image Segmentation, e.g. deleting all blue color, removing background color, or coloring only one segment. This way, not all of the vertices traversed.

Below are some result of author's implementation on some images. Images are intentionally selected to find both power and weakness in author's implementation in cutout filter.

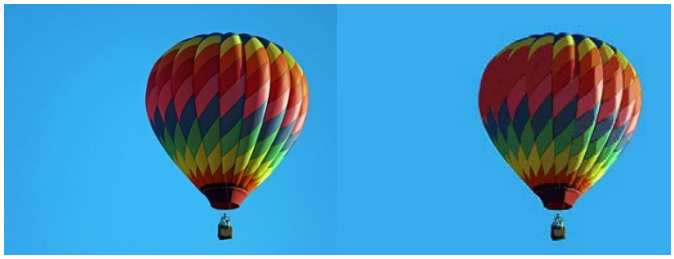

Figure 2. Example of before and after cutout filter based on Image Segmentation with threshold 80 and radius 3. The colors are easy to differentiatte, so it is relative easy to make the cutout.

Figure 3. Example of before and after cutout filter based on Image Segmentation with threshold 80 and radius 3. The result is good enough and not decreasing its artistic value.

[2] https://helpx.adobe.com/photoshop/using/filter-effects-reference.html, retrieved December 3rd, 2017

Figure 4. Example of before and after cutout filter based on Image Segmentation with threshold 60 and radius 3.

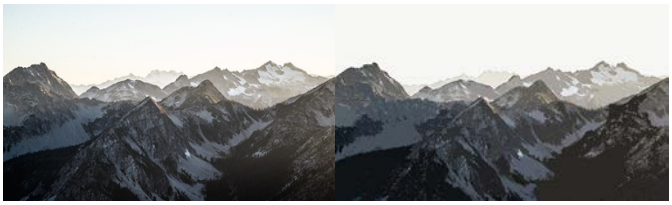

Figure 5. Example of before and after cutout filter based on Image Segmentation with threshold 60 and radius 2. Some of details are disappear, but the simplification of the details make it more interesting.

Figure 6. Example of before and after cutout filter based on Image Segmentation with threshold 60 and radius 3. The result is both better and worse than the Adobe Photoshop's cutout in each of its aspect.

From the result, we can see that many image details are removed. Some of them are good, while some of them ruin the image itself. For example, the details in balloon are removed for good, while details in the person's hair are gone and make the hair weird. In the mountains, white snow make the cutout effect has better result, meanwhile the mountain without snows is not really good compared to the one with snows.

After all, the term good and bad, better and worse, are subjective and can't be defined scientifically. Readers can be the judge for the results itselves. The result may not clear because of the size, so the high resolution image can be found in appendix of this paper, included with author's implementation.

In the experiment, the author intentionally add timestamp for every process. Then, the time consumed for each step can be viewed as seconds. From the shown elapsed time, the code runs pretty fast when making nodes, sorting the edges, searching all the set every nodes belong to, counting average, and creating the output pixels. But, it runs quite slowly when generating edge list and running the Kruskal's MST algorithm. In average, it runs nearly a half minute with < 200,000 pixels and radius value 3 and it runs nearly a quarter minute with < 200,000 pixels and radius value 2. The time grows polynomial to number of pixels and radius. The time complexity is heavily affected by time for generating all possible edges and generating the MST.

Here is the comparison of time elapsed on the same picture, with different radius:

```
[  0.104] 152400 nodes are made
[ 21.007] 5457312 edges made
[ 23.219] edges are sorted
[ 31.131] mst is made with 138481 edges
[ 32.262] 13919 parent listed
[ 32.672] average counted
[ 32.784] replaced
[ 32.841] out pixels are made
```

```
[  0.099] 152400 nodes are made
[  8.160] 2428704 edges made
[  9.294] edges are sorted
[ 13.033] mst is made with 132566 edges
[ 14.088] 19834 parent listed
[ 14.214] average counted
[ 14.325] replaced
[ 14.385] out pixels are made
```

Figure 7. left: execution time for radius value 3, right: execution time forradius value 2. Both run on same picture with same threshold. Note: MST time are sum of sorting and making MST.

Even though we expect Kruskal's (with time complexity O(E log V)) slower than generating edges (with time complexity $O(VR^2)$), we see that the difference between radius value 2 and 3 is much affecting the runtime. In addtion, when program listing all of vertices neighboring the vertex, it loops 4 times for each direction. Because the author use 4 direction system, the loops are increased by coefficient 4. But, in complexity notation, this constant/coefision is removed.

In practice, MST are stopped when minimum edges weigth are more than the threshold. This heavily affect the runtime, then in average case, MST is much faster than generating edge list. Still, the worst case is when all of the pixels have the same color so the big-Oh notation holds $O(\max(E \log V, VR^2))$.

From the complexity, we can see that the algorithm increases heavily when the resolution of the picture is increased. Resizing the picture to become bigger will need much more time to process. In graphic design, we can compress the picture to give a preview first, then actual size will be render after the user satisfied with the result. In Adobe Photoshop CS6, this filter also use many resource and time, so the MST-based cutout filter is not really bad compared to other approach, but still have room for development.

V. CONCLUSION

Image Segmentation is one of image processing step that is very useful for advance image processing application. There are numerous application: face recognition, Optical Character Recognition (OCR), biometrics verification, image recognition, brake light and pedestrian recognition, and many other topics. MST-based clustering is used and discussed in this paper for image segmentation and cutout filter is used for example of its application in graphic design field.

Total complexity of the program is $O(\max(E \log V, VR^2))$. MST-based clustering approach is still one of good and efficient algorithm, but the algorithm still can be developed to be more efficient and giving better result.

In cutout filter, image segmentation can give its best result when the image have many sharp details, but make the image worse when the image have many gradients.

VI. Appendix

The author's implementation of algorithm discussed in this paper can be found in author's github repository (https://github.com/yonasadiel/mst-based-image-segmentation). All of the original and result image have been stored in the repository. It used Python3.6 language with PIL (Python Imaging Library) module. Usage, sample images, and the output from sample images can be found in readme file and image folder.

The sample images are retrieved at December 3rd, 2017 from urls listed:

- house: http://www.palmatin.com/wp-content/uploads/2013/06/painted-square-log-house1.jpg
- mountains: https://static.pexels.com/photos/15382/pexels-photo.jpg
- balloon: https://i.ytimg.com/vi/7_Y3QFdmpHw/maxresdefault.jpg
- flower: https://static.pexels.com/photos/36753/flower-purple-lical-blosso.jpg

The last sample image is photo of the author, therefore the source is not included.

VII. Acknowledgment

The author wans to thank Dr. Judhi Santoso, M.Sc as the lecturer of Discrete Mathematics IF2120 course in author's class. The author also would thank Bandung Institute of Technoloy for its access to IEEE document and papers. Many inspirations and references affect author works. The author would also thank the contributors to open source PIL python library which helped author's implementation of image segmentation algorithm. The author also thanks Jonathan Christopher, Bandung Institute of Technology junior student for the inspiration of this paper.

References

[1] Anandarup Roy, Swapan Kumar Parui, Amitav Paul; Utpal Roy. 2008. "A Color Based Image Segmentation and its Application to Text Segmentation" *Computer Vision, Graphics & Image Processing, 2008. ICVGIP '08. Sixth Indian Conference on* doi: 10.1109/ICVGIP.2008.69

[2] Xue-xi Zhang and Yi-min Yang. (2008). "Minimum Spanning Tree and Color Image Segmentation" *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on* doi:10.1109/ICNSC.2008.4525344

[3] K.H. Rosen, *Discrete Mathematics and Its Application*, 7th ed. New York: McGraw-Hill, 2012, pp. 641-802.

[4] S. Halim and F. Halim, *Competitive Programming 3*. Singapore: Lulu, 2013, pp 49-53 and 121-139.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017

Ttd

Yonas Adiel Wiguna
13516030