Minimum Spanning Tree-based Image Segmentation and Its Application for Background Separation

Jonathan Christopher - 13515001
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
jonathan.christopher@students.itb.ac.id

***Abstract*—It is often needed to separate the background of images from foreground objects, which might not always be uniformly colored. This paper presents a minimum spanning tree-based algorithm which treats the image as a graph and generates a minimum spanning forest in which each minimum spanning tree is a region of the image, using a modified version of Kruskal's algorithm. The colors in each region is then averaged, and a region is selected as the background. This algorithm generally runs in linearithmic (*O(n log n)*) time. This paper also considers the application of this algorithm for smoothing scanned document images.**

***Keywords*—minimum spanning tree, Kruskal's algorithm, image segmentation, background separation.**

I. INTRODUCTION

Advances in fields such as image processing and digital publishing has made it possible to use digital images in new ways. A digital image can now be cropped and shaped according to needs, using various image editing tools. Although most available tools for image editing do help significantly in editing only specific regions of an image, they still require manual human operation. It is because the human mind is still superior to machines in recognizing and separating objects in a flat image, due to the ability of the mind to learn and reason about the objects presented in the image themselves.

However, sometimes it is needed to separate and remove backgrounds autonomously, for example to for batch processing a large number of images, or for real-time uses such as the preprocessing step of a video object tracking system. This requires an adaptive algorithm that is able to efficiently process the image, using only the limited color data found in a typical digital image file.

This paper explores the properties and possible applications of a minimum spanning tree-based image segmentation algorithm for background separation. An algorithm utilizing a modified version of Kruskal's minimum spanning tree algorithm is implemented in Python and tested with several sample images.

The algorithm discussed in this paper is a graph-based algorithm for image segmentation outlined in [1], with several modifications meant to simplify its implementation and improve its effectiveness in handling various types of images. It is also inspired by the minimum spanning tree-based hierarchical clustering algorithm in [2], as the problem of separating regions in an image can be reformulated as the problem of clustering pixels based on their color and positions.

II. GRAPHS, TREES AND MINIMUM SPANNING TREES

A. Graphs

A graph $G = (V, E)$ consists of a nonempty set of vertices/nodes V, and a set of edges E, each of which connects two endpoint vertices together [3]. A graph may have no edges (empty/null graph), or have each of its vertices connected by an edge directly to each other (complete graph). The edges in a graph may be weighted, which means they are assigned a cost/weight value. Graphs are frequently used to mathematically model relations between objects, such as roads and cities in maps, states and transitions in a finite state automaton, or computers in a network.

Two vertices on a graph are adjacent if there is an edge connecting them. The edge is then incident to each of the vertices it is connected to. The degree of a vertex $\deg(v)$ is the number of edges incident to it, except for loops (edges which two ends are connected to the same vertex), which are counted twice. The neighbors of a vertex is the set of vertices which are adjacent to it [3].

There are several types of graphs, grouped by the directedness of their edges, the existence of loops and whether they have multiple edges. A simple graph has undirected edges, single edges only and no loops. A multigraph has undirected, possibly multiple edges and no loops. A pseudograph is like a multigraph, but with loops. A simple directed graph has single, but directed edges, and no loops. A directed multigraph has directed, possibly multiple edges, with no loops. A mixed graph may have single or multiple edges, which might be directed or undirected, and includes loops [3].

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is a graph which set of vertices V' is a subset of V, and which set of edges E' is a subset of E and consists only of edges where both incident vertices are members of V`. A subgraph can be constructed from a graph by removing edges, or by removing vertices and their incident edges [3].

A path is a sequence of edges beginning at a vertex, connecting vertices along. Formally, it is a sequence of edges in a graph such that there is a sequence of vertices where the i^{th} edge in the sequence connects the $(i-1)^{th}$ and i^{th} vertex. A circuit/cycle is a path that starts and ends at the same vertex. A simple path does not contain the same edge more than once. In a simple graph, the sequence of vertices passed through by a path is enough to uniquely denote the path, as there are only zero or one possible edge between each vertex [3].

B. Trees

A tree is defined to be a connected, undirected graph which contains no simple circuits. As a tree does not contain simple circuits, it must not have a loop or multiple edges. Thus, a tree is a simple graph. Each vertex in a tree has a simple path connecting it with all other vertices. A forest is a collection of trees [3]. Trees are commonly used to model hierarchical relations, for example in family trees and organizational trees. Trees can also be used to structure data into forms more suitable for efficient processing, such as a binary search trees or a disjoint set union/find data structure, the latter which will be utilized in this paper.

A tree has several properties. There are exactly $n-1$ edges in a tree of n edges. A tree may be rooted, in which a vertex is designated as the root and every other vertex is directed away from it. Sometimes, the directedness is not explicitly shown. Vertices which are adjacent to a vertex, in the direction of the edges, are the children of the vertex. A vertex that has no children is a leaf; a vertex that has one or more children are an internal node. An m-ary tree is a rooted tree which internal vertices has at most m children. The height of an m-ary tree is the length of the longest path from the root to a leaf. In an m-ary tree of height h, there are at most m^h leaves [3]. As in a graph, the edges of a tree may be weighted or unweighted.

A spanning tree of a simple, connected graph is its subgraph, which is also a tree and contains all its vertices [3]. Any simple graph which is not yet a spanning tree can be made into one by removing edges that make a cycle. Only connected simple graphs have spanning trees, and vice versa. Removing an edge from a spanning tree will split it into two spanning trees. Traversing the vertices of a connected graph can be done according to one of its spanning trees, either through depth-first search or breadth-first search.

C. Minimum Spanning Trees

A minimum spanning tree of a connected weighted graph is a spanning tree with the smallest possible sum of edge weights [3]. There can be multiple minimum spanning trees if the weights of a graph's edges are not unique. Minimum spanning trees, or MSTs, are useful in modelling problems where adding edges increases costs, such as finding the least expensive way to connect computers in a network. Several algorithms to determine the minimum spanning tree of a graph exists; two of the most efficient and easiest to implement are Prim's algorithm and Kruskal's algorithm.

In Prim's algorithm, we begin by taking a starting vertex. For another $n-1$ iterations or until we run out of vertices to visit, we then take a vertex which is adjacent to a vertex already taken, which edge connecting it to the taken vertex is minimum among all currently available edges, but will not form a cycle if taken. While doing this, we keep track of which edges are used and the sum of their weights. This will result in a minimum spanning tree of the graph which contains those edges, if the original graph is connected. If the original graph is not connected, the algorithm will only find a minimum spanning tree for the connected component which contains the starting vertex in most implementations.

An efficient way to implement Prim's algorithm involves a priority queue to keep track of currently available unvisited incident vertices, which are vertices which has an edge adjacent to one of the currently taken vertices. The lower the weight of the connecting edge, the higher the priority. The complexity of an efficient priority queue implementation can be as low as $log(n)$, for instance if using a self-balancing binary search tree-based priority queue implementation. To prevent creating a cycle, we keep track of which vertices are already taken after each iteration. This is commonly done using an array of Boolean values, which could be read/written in $O(1)$ time. As the process is iterated n times, the overall time complexity is $O(n \log n)$ [4].

In Kruskal's algorithm, the edges of the original graph are first sorted according to edge weight. The new minimum spanning tree is first an empty graph consisting of the vertices of the original graph. We then try to add $n-1$ edges which does not form a cycle, starting from the edge with the least weight in the sorted edge list. When $n-1$ non-cycle-forming edges has been added, the minimum spanning tree is constructed.

To efficiently implement Kruskal's algorithm, we need to use a highly efficient, tree-based data structure called disjoint set union/find to check whether an edge will form a cycle if added. The time complexity of an merge/query operation of this data structure approaches $O(1)$. The sorting of the edges itself is bound by the complexity of the sorting algorithm uses. In most general cases, this will be $O(n \log n)$, but in more specific cases where a faster sort algorithm can be used (for example counting sort), it might be improved to $O(n)$. Overall, the total time complexity of a general-purpose implementation is $O(n \log n)$, similar to Prim's algorithm [4].

The union/find disjoint set data structure is a tree-based data structure used to check whether two elements belong to the same set. Elements are represented as vertices, which parent vertex is also stored. Elements which are in the same set are part of the same tree. Initially, all vertices are disconnected (the parent of each vertex is itself). A query can be made to determine the root of a vertex, by recursively checking its parent vertex. While performing the recursive query on a vertex, the corresponding vertex is also moved upwards towards the root. This ensures that the height of each tree is at most two, which means that a query can be executed in near *O(1)* time. When a merge operation is performed on two vertices, one of the vertices will be designated as the child of the other's root. Checking whether two vertices are in the same set is then simply a matter of checking whether both vertices have the same root [4]. It is used in the implementation of Kruskal's algorithm to check whether both vertices of a new edge are already connected; if yes, then the edge should not be added as adding it will create a cycle.

III. Image Background Separation

The basic idea for recognizing and separating an image's background is to first split it into regions. A region or segment of an image must fulfill several characteristics. Pixels in the same segment must be near in position, but also of similar colors. These characteristics alone are far from optimal, as the segmentation of images also depends on the objects in the image; however, the recognition of objects is beyond the scope of this paper, and thus we must work with these characteristics only.

To separate pixels according to the criteria above, we must first transform the image into a graph representation. Each pixel in the image is regarded as a vertex. Each pixel is mapped to a five-dimensional space, composed by their vertical and horizontal positions, and three color components. Pixels are connected to their neighbors by edges, weighted by a distance function which takes both pixels' relative position and difference in intensity or color.

The graph will then be processed and split into multiple minimum spanning trees, according to the criteria above. The splitting process utilizes a modified version of Kruskal's algorithm. Each tree represents a part of the image; vertices which are part of a tree are the vertices which make up a segment. Only edges with the least weight are kept, which means that only pixels that are similar will be connected in the same segment.

```
for p in pixels
  add edge between p and
  neighboring pixels to edgelist

sort edgelist

initialize segments with pixels

for e in edgelist
  if v1, v2 in e not yet connected
    merge v1, v2 in segments

background = largest s in segments
```

Figure 1. Pseudocode for the background separation algorithm

A problem that arises when processing images is the comparatively large amount of data that must be processed. A typical medium-resolution image has a size ranging between 200x200 pixels to 1000x1000 pixels; high resolution images that are common nowadays has even greater pixel counts. This meant that an image processing algorithm must be able to efficiently process millions of pixels. The time complexity required to finish executing the algorithm in reasonable time with the typical computer specifications of today is approximately *O(n)*, with *O(n log n)* barely able to cope.

Ideally, the algorithm would start by generating a complete graph of all pixels. However, this requires $O(n^2)$ time to generate the edges, as a complete graph has $\frac{n(n+1)}{2}$ edges. It would then require too much time to process even medium-resolution images. Instead, an edge are created only if its two pixels are located closer than or equal to an outer radius parameter. When a small value for outer radius is used, this reduced the complexity of edge generation to near *O(n)*, as every pixel would only have a constant amount of edges associated to it.

An example of a distance/difference function used to weight the edges of the image graph is a function that returns the Euclidean difference between two pixels' colors:

$$d = c_R^2 + c_G^2 + c_B^2 \qquad (1)$$

A better distance function would also need to take into account both pixels' relative positions. However, as we only connect pixels in a radius, the relative position differences are already implicitly considered.

Next, we run a modified version of Kruskal's algorithm on the generated image graph. The list of edges between pixels is sorted in ascending order, based on the edges' assigned weights. Edges are then added from the list to the graph, such that only edges that does not create a cycle are allowed to be added. Specifically, the only edges that may be added are edges which does not connect two vertices that are already connected, whether directly or indirectly. Connected vertices represent a segment in the image; initially all pixels are individual segments, but over time, similar pixels are joined together. The connectivity check is performed using a query on a union/find disjoint set data structure. However, the iteration condition is modified: edges will be added only until the weight of the currently processed edge exceeds a specified threshold. The time complexity of this step is bounded primarily by the sorting algorithm used; general sorting algorithms typically uses *O(n log n)* time. If the edge weights are discrete and its range is small, a more efficient sorting algorithm such as counting sort or radix sort may be used, improving the time complexity to *O(n)*.

The minimum spanning trees formed by the modified Kruskal's algorithm represent the segments of the input image. The total difference between each pixel in each segment is minimum, and is at most equal to the threshold parameter value. Edges with weights larger than the threshold are not joined. This ensures that regions of the image which has contrasting colors are not joined together. As the possible edges generated can only connect pixels not greater than the outer radius parameter in distance, regions will also be close together in position, even mostly contiguous.

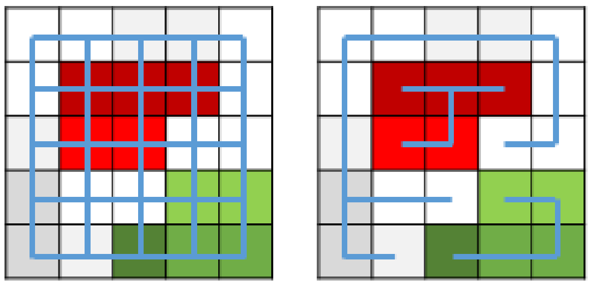


Figure 2. The image is first transformed into a graph, with edges connecting neighboring pixels closer than the outer radius parameter (left). The graph will then be separated into multiple minimum spanning trees, with each minimum spanning tree representing a segment in the image (right).

This algorithm will join regions with smooth color gradients, as the color differences between adjacent pixels in smooth gradients are small. However, this property causes problems when trying to process blurry images, as soft region edges may cause the region to be joined with the background. An obvious solution is to preprocess the image with a sharpening filter, though experiments shows that sharpening filters increase the amount of noise in the image, thus lowering its quality. An alternative solution is to define an inner radius parameter, which prevents the algorithm from creating edges which connects pixels that are too close together, so that region edges can be distinguished more accurately.

The number of separate minimum spanning trees or image segments t can be calculated from the number of pixels/vertices v and the number of edges e used to create the minimum spanning trees:

$$t = v - e \qquad (2)$$

From these t segments, we need to select one which represents the background of the image. The simplest method, which is used in the author's implementation of this algorithm, is to select the segment with the most pixels. Alternatives exist, for example to select segments with sizes exceeding a limit or segments which are located near the borders of the image. The segments are then marked by changing the color of pixels contained in each segment. In the author's implementation, the background segment is colored white, while the other segments are set to the average color of all pixels in each segment.

The total time complexity of this algorithm is generally *O(n log n)*, not much different Kruskal's algorithm. The most time-consuming processes are the edge creation (*O(log n)*, but with a large constant factor depending on the radius parameters) and the edge sort *(O(n log n)*). In certain cases, however, the sorting algorithm could be replaced with an *O(n)* algorithm such as counting sort, so that the overall time complexity can be reduced to *O(n)*.

IV. Applications

This algorithm can be used to automatically remove the backgrounds of images. However, because of its limitations, the author's implementation of the algorithm will only work best on images with clearly defined backgrounds.

This algorithm can also be applied to correct photos of documents, i.e. to remove shadows and lighting effects. A document photo typically contains black text or figures on a white background, which has high contrast and easy to distinguish from the text. An interesting side effect is to slightly reduce noise, as pixels are grouped together in the segmentation process.

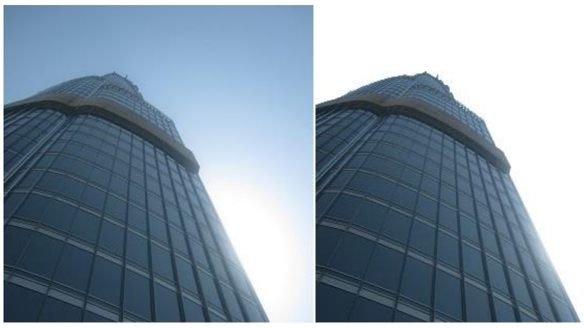
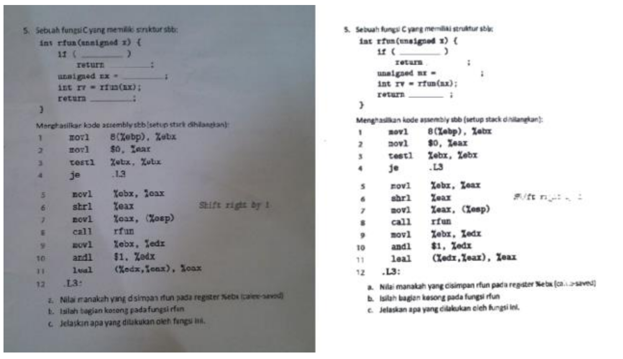


Figure 3. Result of algorithm (threshold = 200, inner radius = 0, outer radius = 3, join foreground = false). Sample image taken by author.


5. Sebuah fungsi C yang memiliki struktur sbb:

```
int rfun(unsigned x) {
    if ( ________ )
        return ________ ;
    unsigned nx = ________ ;
    int rv = rfun(nx);
    return ________ ;
}
```

Menghasilkan kode assembly sbb (setup stack dihilangkan):

```
        movl    8(%ebp), %ebx
        movl    $0, %eax
        testl   %ebx, %ebx
        je      .L3

        movl    %ebx, %eax
        shrl    %eax              Shift right by 1
        movl    %eax, (%esp)
        call    rfun
        movl    %ebx, %edx
        andl    $1, %edx
        leal    (%edx,%eax), %eax
    .L3:
```

a. Nilai manakah yang disimpan rfun pada register %ebx (callee-saved)
b. Isilah bagian kosong pada fungsi rfun
c. Jelaskan apa yang dilakukan oleh fungsi ini.

Figure 4. Result of algorithm for correcting photo of document (threshold = 200, inner radius = 0, outer radius = 3, join foreground = true). Sample image taken by author.

To prevent segments inside closed shapes being treated separately from the background (such as segments inside the letter 'o', 'p', 'd', and such), we set the outer radius parameter to several pixels wide. To handle slightly blurry images, we increase the inner radius parameter. However, it should be kept in mind that the larger the difference between the outer radius and inner radius parameter, the slower the execution time will be, as more edges will be considered.

V. Conclusion

Recognizing image backgrounds is a useful problem in image processing, which may be applied for automatic background removal or scanned documents color correction. An algorithm for separating image backgrounds based on minimum spanning trees is discussed and implemented in this paper. However, the current implementation leaves much room for improvement, both to the quality of the result and to its efficiency.

VI. Appendix

The author's implementation of the algorithm discussed in this paper and the sample images shown can be accessed on Github (https://github.com/nathanchrs/mstsegment). It is written in Python, and requires the Pillow image processing module and NumPy scientific computing module to run.

VII. Acknowledgment

The author thanks Dra. Harlili S. M.Sc. as the lecturer of the author's Discrete Mathematics class, for guidance in preparing this paper. The author would also like to thank the developers and contributors of the open-source Pillow Python image library and NumPy scientific computing library, which has greatly simplified the author's implementation of the image segmentation and background separation algorithm.

References

[1] P.F. Felzenszwalb and D.P. Huttenlocher, *Efficient Graph-Based Image Segmentation*. [Online]. Available: http://cs.brown.edu/~pff/papers/seg-ijcv.pdf (Retrieved 30 November 2016).

[2] M. Yu, *et al*. (11 February 2015). *Hierarchical clustering in minimum spanning trees*. [Online]. Available: https://www.nas.ewi.tudelft.nl/people/Piet/papers/Chaos2015_Hierarchical_clustering_MSTs.pdf (Retrieved 4 December 2016).

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

[4] S. Halim and F. Halim, *Competitive Programming 3*. Singapore: Lulu, 2013, pp. 52-53 and 138-144.

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, 9 Desember 2016

ttd.

[illegible]

Jonathan Christopher - 13515001