Image Foreground/Background Segmentation Using Max Flow Algorithms

Chathura Abeyrathne • chath@bu.edu
Raj Vipani • rvipani@bu.edu
Sofia Briquet • sbriquet@bu.edu

I. Problem Description

Image Segmentation is a topic in computer vision with applications ranging from image augmentation to object classification and also in medical imaging. The problem of accurate labeling and grouping of pixels can thus be used for a variety of tasks. The objective of this project is to separate the foreground and background of an image. To achieve this we use the concept of the min-cut max-flow theorem and implement the Edmond-Karp algorithm and the Push-Relabel algorithm to get a segmentation of the input image. We also compare our results with the k-means clustering algorithm.

II. Background

The basis of our approach relies on network flow graphs, which are directed graphs where each edge has a capacity that amounts to the flow that can be pushed through. Generally, there are two distinguished nodes, the source and the sink, where flow comes from and goes to, respectively. Then, the max-flow theorem needs to be applied to the network flow graph to measure the maximum flow between the source node and the sink node. The min-cut max-flow states that the value of the minimum cut is the same as the value of the maximum flow. So the minimum cut problem is to find the set of edges with the smallest sum of capacities that need to be removed in order to separate the source node and the sink node, which will in turn equal to the maximum flow that can be pushed between these two nodes. The max flow/ min-cut must be calculated while satisfying the following two constraints:

- *Capacity Constraint* - This constraint states that for each edge $(u,v) \in E$, there is a non-negative value known as the capacity of the edge $c(u,v)$ associated with it and that the flow passing through the edge must be less than or equal to the edge's capacity.
- *Conservation of flow* - This constraint states that for every vertex in the network, the amount of inflow must be equal to the amount of outflow.

Another important concept for our approach is residual graphs, which have all of the same edges as the original network flow graph but a new edge is added in the opposite direction for every edge with a capacity equal to zero. An augmenting path is a path from the source to the sink that only goes through edges with positive capacities, so whenever a new augmenting path is found, the total flow of the graph can be increased. To keep increasing the flow of the graph, new augmenting paths need to be found while decreasing the capacity of the edges the path goes through. To maintain flow conservation, whenever the capacity of an edge is decreased, the residual edge needs to be increased by the same amount. Hence, if no more augmenting paths can be found, the maximum flow has been discovered, and at the same time, in the residual graph, the minimum cut has been found since a path from the source to the sink doesn't exist. Lastly, the maximum flow found with the max-flow algorithms which represents the cut between source and sink, is used as the boundary between the two regions, eg. the foreground and the background.

III. Approach

The first task of our project is to convert a grayscale image into a network flow graph. The way we decided to implement the flow graph was to define each node with two values corresponding to the x and y coordinates of each pixel. Then 2 more nodes were added, one as the source node, and a sink node, both implemented as 2d arrays as well. Lastly, the graph was defined to be a vector of a map of nodes and the edges. The edges are defined as the neighbor pixels and the weight (capacity) of each arc was defined with a function of similarity and intensity, so for example two foreground pixels will be more similar to each other, hence a higher edge weight of the arc joining them.

The max-flow algorithm chosen were the Edmonds-Karp algorithm which uses the pretty common Ford-Fulkerson max-flow method and the Breadth-First Search algorithm to find augmented paths in the graph and then relies on residual graphs to keep increasing the flow between the source node and the sink while maintaining flow conservation for every node other than the source and the sink. Although the Edmonds-Karp is a popular algorithm to find the maximum flows in a graph, it is not very fast, that is why we also chose to also implement the Push-Relabel algorithm $O(EV^2)$ which is generally faster than the Edmond-Karp algorithm $O(E^2V)$. Similar to the Edmonds-Karp algorithm, the Push-Relabel algorithm also works on augmenting paths but defines a node to be overflown to then push the highest amount of flow possible between the sink and source. So even though the two algorithms work differently, they should both find the same maximum flow which would correspond to the minimum cut to segment the desired image.

The main focus of this project is to find the optimal background/foreground segmentation of an image using network flow graphs and max-flow algorithms. Nonetheless, we decided to also include the machine learning K-means algorithm, which is a common and fast clustering algorithm that defines K number of means and then labels each pixel as foreground or background depending on how close the point is to each of the two means. Clustering pixels is a common approach to the problem of

image segmentation or object recognition so we decided to include in the project for comparison of the output image between segmentation produced by max-flow and segmentation produced by clustering.

Lastly, we also set up our project to be user-interactive, with a python GUI where the user selects the input grayscale image and then needs to define one pixel (point in the image) as the background and one as the foreground so the algorithm that converts the image into a network flow graph knows which pixel to define as the source node, which corresponds to the background, and which pixel to define as the sink or the foreground reference. The user can then decide which of the three mentioned algorithms to call to segment the image. All algorithms were implemented in c++ and then integrated into a python framework.

IV. Implementation

Grayscale Square Images into Network Flow Graphs: From the literature review [1], we found an implementation to convert an m*n grayscale image into a network flow graph. Each pixel in this image is considered as a node on the graph. Thus in total, we have m x n number of nodes. Additionally, we have 2 more nodes which are referred to as the source and sink nodes. For the edges, there are two types of edges in the graph. Type 1 connects each pixel to the source and to the sink while Type 2 which are inter-pixel edges that connect each pixel to its 4 neighboring nodes from the original image.

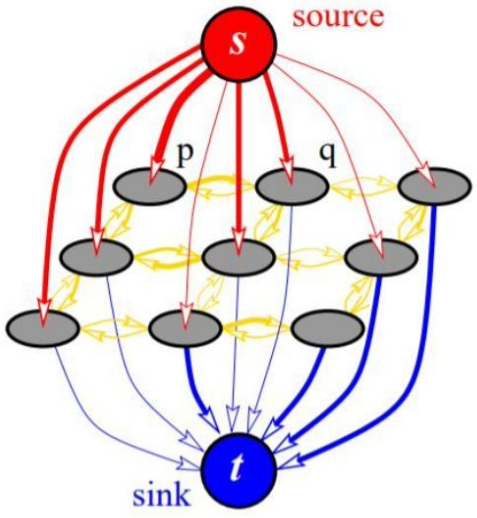


Figure 1: Visualization of converting a 3*3 image into a graph and the connecting source to pixel, pixel to sink and inter pixel edges

The weights for these edges which also is the capacity of the edges are calculated by comparing the intensity value of the pixels which these nodes correspond to. The idea behind it is that if two connected pixels are identical, that is they belong to the same segment, the edge weight should be

higher and respectively smaller for pixels that are not identical to each other. The weights between two edges are calculated by the formula:

$$W_{u,v} = 100.\, exp(-\frac{(I_p - I_q)^2}{2\sigma^2})$$

Here, I_p and I_q are the intensities of pixels p and q respectively and σ from empirical results was chosen to be 30.

Edmonds-Karp Algorithm: The Edmonds-Karp algorithm is an implementation of the Ford Fulkerson algorithm. While the Ford Fulkerson method states that if there exists a path between the source and the sink in the residual graph, we can augment the current flow through this path. The Edmonds-Karp algorithm extends this idea and says that a path from the source to sink in the residual graph can be found using a Breadth-First Search Implementation. The pseudo-code for the Edmonds-Karp algorithm is as follows:

```
Inputs
      residual_graph        : Residual Graph obtained from flow graph
      src                   : Source
      target                : sink or target

Output
      maxFlow               : Maximum Flow Rate

The Edmonds-Karp Algorithm:
      maxFlow = 0             // Initialize the flow to 0

      While true:
            path = BFS(residual_graph, source, sink)

            if path.length() == 0:
                  break

            current_flow = min(residual capacity of all edges along this path)

            for every edge (u,v) along the path:
                  residual_capacity(u,v) -= flow
                  residual_capacity(v,v) += flow
            maxFlow += flow
```

Figure 2: The Edmonds-Karp algorithm runs in $O(VE^2)$ time.

Push-Relabel Algorithm: this max-flow algorithm was implemented in c++. It first initializes a preflow, which is defined to be a flow through the graph that doesn't necessarily satisfy the flow conservation constraint. Hence, the flow entering a node can exceed the flow leaving the node and excess flow occurs at an overflown vertex. Each node in the graph is augmented by a height feature which determines how the flow can be pushed, which can be solely done from a higher to a lower node.

The algorithm was initialized by saturating the edges from the source node and setting the height of the source to equal the number of nodes in the graph which is m x n. The height of all other nodes was set to zero. The algorithm then works in a while loop until there are no more overflown vertices. For a given node *u*, the algorithm can push flow through it if a non-saturated edge is found, the pushed flow is equal to the minimum quantity between the residual capacity of the edge and the excess flow of u. So once the flow is pushed, the excess flow at *u* falls by this amount while the excess flow of *v* increases by the same. The relabel part of the algorithm comes when flow cannot be pushed through an edge, the algorithm then *relabels* a node by changing the height of said node to be equal to $1 + min(height(u), height(v))$. As mentioned before, push-relabel should generally run faster than Edmonds-Karp since in a network flow graph there is a higher probability to have more edges than nodes, and the complexity of push-relabel depends on the number of nodes squared in contrast to Edmond-Karp where complexity depends on the number of edges squared. To call the algorithm:

```
Int max_flow = PushRelabel(source, sink)
```

K-Means Algorithm: The k-means algorithm was also implemented in c++ using source code found online and then adapted to work with the input images which are read within the function as a matrix and then k means is called to return the labels of each pixel which can either be 1 or 0 depending on how the unsupervised learning algorithm classified the background and foreground points. The running time of K-means is O(V^2), so as it depends solely on the total number of pixels, which translates to the total m x n nodes of the image, it should run considerably faster than any of the max-flow algorithms mentioned above.

Python GUI: As mentioned above, all the algorithms in the project were implemented using C++. Python tkinter package was used for the frontend Graphical User Interface (GUI) due to its simplicity and readily available documentation as well as community support. Python ctypes library was used to create an interface between Python frontend and C++ backend. All three algorithm implementations which are in C++ were called in three separate C wrapper functions. This is because ctypes Python lib only supports calling C functions. Then C functions with C++ source codes were compiled into shared libraries using g++. Upon executing the Python script responsible for GUI, compiled shared libraries will be loaded by ctypes package, and backend C++ functions will be called as per the instructions coming from the GUI.

Figure 3 shows the dataflow of the project from the Python-based GUI/frontend to C++ based backend and packages used for each communication link.

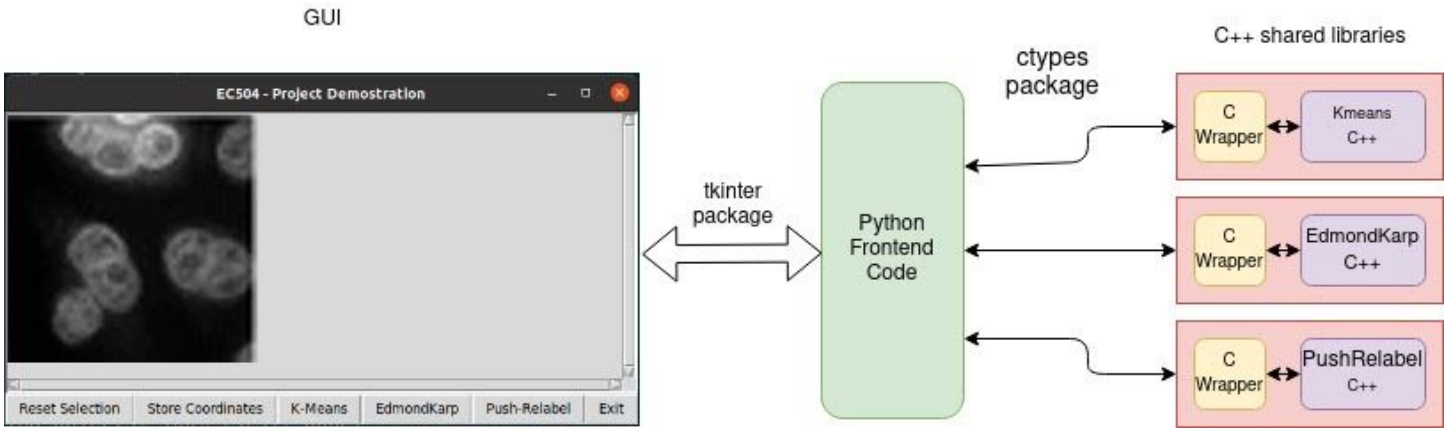


Figure 3: Dataflow of the project from the Python frontend to C++ backend

V. Code

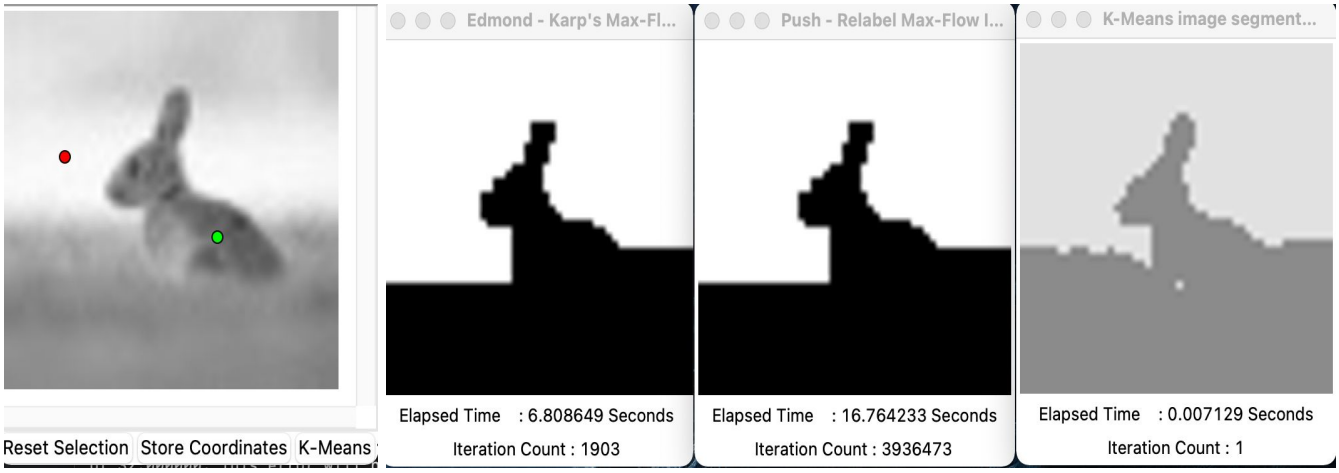
GITHUB Link: https://github.com/rajvipani/Image-Segmentation-Using-Max-Flow

How to Run the Code: The code was all integrated in a makefile, including a folder of test images. To compile the programs the user only needs to call *make* from the terminal and then *make run* to call the GUI and then proceed with the image segmentation.

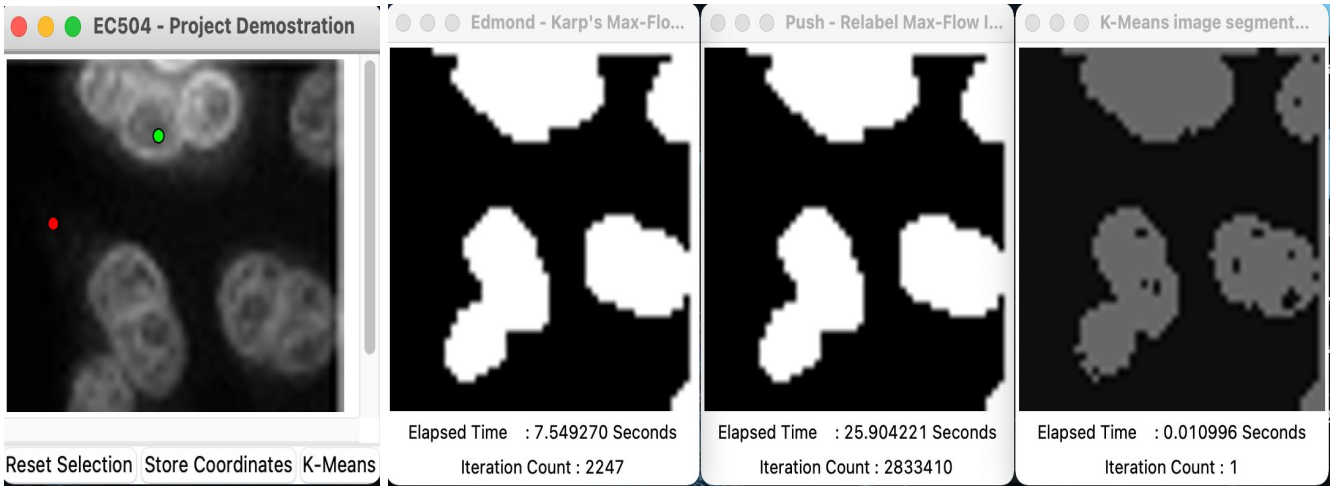
Since the GUI uses several python packages and opencv in both the c++ backend and python3 frontend, the user may need to download the required packages by calling *apt install python3 python3-pip* to download python3 and then *pip3 install -r requirements.txt* to download python requirements. Lastly, the user may need to also call *apt install libopencv-dev* to download the c++ opencv library. Detailed instructions for each step are provided in the Github repository readme file. After every program execution, the segmented images of the current run are saved in the GUI folder for reference.

VI. Results and Discussion

Upon implementation of the code, we tested it for a set of images. Attached below are examples of the obtained results along with the number of iterations each algorithm took and the execution times. The obtained segments are for the two foreground and background points chosen from the UI represented by the green and red dots respectively. Since the number of nodes on the flow network depends on the number of pixels, convergence takes a long time for larger images. For testing purposes, we rescaled the input images to a 50 * 50 size.

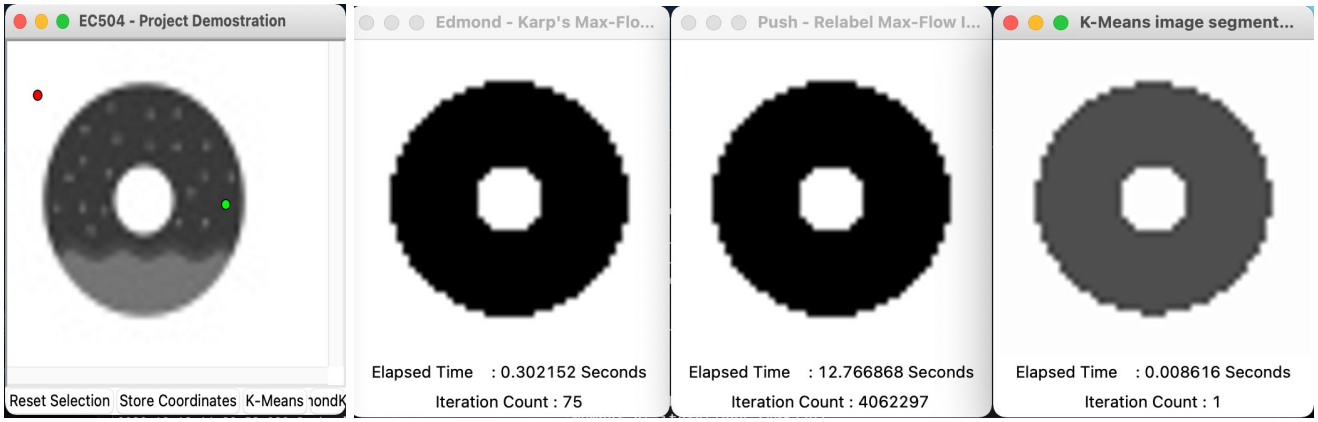


(a) (b) (c) (d)



(a) (b) (c) (d)

(a) (b) (c) (d)



(a) (b) (c) (d)

Fig: [left to right] a. Displaying input image and selecting the foreground and background points
b. Segmentation obtained from the Edmonds-Karp algorithm
c. Segmentation obtained from the Push Relabel algorithm
d. Segmentation obtained from the K means clustering algorithm

VII. Work Breakdown

The image to graph algorithm was implemented by Raj Vipani as well as the Edmonds-Karp max-flow algorithm. Sofia Briquet implemented the Push-Relabel algorithm which was later adapted by Raj Vipani to work with the definition of the image graph already created. Chathura Abeyrathne worked on the python front-end part of the project as well as integrating all of the back-end algorithms into the python GUI. Chathura also utilized a K-means clustering code to segment images and compare the results with the max-flow algorithms.

The presentation slides and the project report was then written by all three members of our team.

VIII. References

[1] Boykov, Y. and Funka-Lea, G., 2006. Graph cuts and efficient ND image segmentation. *International journal of computer vision*, *70*(2), pp.109-131.

[2] http://www.cse.unsw.edu.au/~cs4128/18s1/lectures/10-network-flow.pdf

[3] https://sandipanweb.wordpress.com/2018/02/11/interactive-image-segmentation-with-graph-cut/

[4] https://julie-jiang.github.io/image-segmentation