Project 1: Image Segmentation

Heather Michaud, Bharath Bogadamidi, Drew Guarnera

Image segmentation itself is a fairly straightforward problem. The basic principle is to group related pixel data from an image that constitutes the background of the image and partition, or segment, those elements from data that makes up the foreground content. However, the actual implementation of the process requires the combination of a few different algorithms working in tandem to produce the desired results. The following describes the specific approach to the image segmentation problem, including the implementation, analysis, and results. The breadth first search and Ford-Fulkerson algorithm were utilized to solve the image segmentation problem, and various optimizations were performed in order to reduce the time complexity. Difficulties encountered with implementation are also discussed, as well as the efficiency, modularity, and generalizability of the implementation. We conclude with the final results of our implementation.

Algorithms

The approach taken for solving the image segmentation problem is the Edmonds-Karp algorithm, which is an amalgamation of two primary algorithms. The first algorithm is a standard breadth first graph search which is utilized to traverse a graph beginning from a given starting point known as the source node and trace the shortest path to a terminating end point referred to as the sink. Our implementation of this module consists of input validation performed by returning invalid results in the form of an empty shortest path list and infinite minimum capacity if the source or sink nodes are invalid. If the nodes are valid, then the shortest path is measured by the least number of edges. This is accomplished by creating a queue that contains the list of nodes that require visitation by the breadth first search, beginning with the sink node. Two lists were maintained throughout the algorithm. One contains the preceding nodes via shortest path for each node in the graph, excluding the starting point. The second list contains the edge weights within the shortest path. Both lists are maintained throughout the execution of the algorithm. While there are still nodes contained within our queue of nodes to be visited, the first node in the queue is popped off, and that node's neighbors that have not already been visited are added to the end of the queue. For each neighboring node found, the neighboring node's identification number is added to our list of the shortest distance predecessors. The weight from the original node to the neighboring node is also recorded at this time. Once the end node has been visited, the list of predecessor nodes is traversed to find the predecessor nodes of the end node. The node identification numbers are stored in a list and paired with the minimum edge weights associated with the shortest path. The resulting pair is returned and used by the Ford-Fulkerson algorithm, which is the second algorithmic component in the image segmentation solution.

The Ford-Fulkerson algorithm is designed to discover the maximum flow and minimum cut of a graph. The process is to take a given path (in our case, the shortest path provided by the breadth first search) and use the maximum flow (which is the bottleneck, or smallest weight) of that path from the source node to the sink node. This is used to create a residual graph in which edge weights can be augmented. Each time the residual graph is updated, the edge in a path with the maximum flow is reversed, which severs it from the set of nodes attached to the sink, and it maintains connection to the set of nodes attached to the source. Each iteration of Ford-Fulkerson is assured to eliminate at least one edge path from source to sink until eventually no paths exist. If no path exists from the source to the sink, a minimum cut has been achieved and the maximum flow values for each path found during the algorithm are accumulated to provide the maximum flow for the entire graph. It is at this time that the algorithm is complete. In relation to solving our image segmentation problem, the residual graph is of most interest to our approach.

The image segmentation solver created for this project is actually a combination of algorithms. Its input is the file path to a portable gray map (.pgm) file. Its output is the same provided image with only the foreground preserved and the background removed. The image contained within the PGM is first converted to a graph. This is accomplished by representing each pixel as a node in the graph. Each pixel

is connected to its surrounding pixels. Thus, each node that represents a pixel is connected to at most four neighboring nodes (in the case that it is located within the middle of the graph) and at least two neighboring nodes (in the case that it is one of the four corner pixels). A super source and super sink node are also added. The super source is connected to every pixel, and every pixel is connected to the super sink. The edge weights for each of the connecting node pairs *u,v* are calculated as follows:

$$w(u,v) = p(t) - |p(u) - p(v)|$$

where *p(t)* is the node value of sink *t*, p(*u)* is the gray scale pixel value of node *u*, and *p(v)* is the gray scale pixel value of node *v*. The source *s* is assigned a node value of 0, which is the minimum value a gray scale pixel can have, and the sink *t* is assigned the maximum value that the provided gray scale image has, as given within the first few lines of the PGM file. Prior to graph creation, the average pixel value is calculated and used as a threshold. During graph creation, while the neighboring nodes of each pixel are traversed, the calculated edge weight is compared against the threshold. If it exceeds the threshold value, the edge is added to the graph, otherwise the connecting edge is not added. Thus, after the graph has been created, the source *s* is connected to all of the pixels associated, to some threshold, with the minimum gray scale value, and all of the pixels associated, to some threshold, with the maximum gray scale value are connected to the sink *t*. In this sense, the source is connected to all nodes within the foreground, and all other nodes which thus must be connected to the sink are considered within the background. The Ford-Fulkerson algorithm is executed once after graph creation with the super source and super sink as inputs to validate that the minimum cut was found. Finally, the output cut image is obtained by keeping the original pixel values of the foreground nodes and overwriting the pixel values of the background nodes with the maximum gray scale value.

Data Structures

The most basic unit of detail regarding the image segmentation solver is how graphs are represented. A graph can be represented using an adjacency matrix or an adjacency list. We chose an adjacency list because it is much more memory efficient for sparse graphs due to the fact that it only keeps track of the existing edges. While the adjacency matrix has a memory complexity of $O(V^2)$, the adjacency list has a memory complexity of O(E). In order to represent the adjacency list, a map of maps from the standard library was used. Maps were specifically chosen over any other data structure for their constant access time. As opposed to a linked list implementation, which would require at worst O(n) for each traversal to find if a neighboring node existed, map access and finding if a key exists is always O(1) and the only penalty is some overhead for hashing calculations. A set was used to easily keep track of the number of nodes within a graph. Sets offer the unique and rapid implementation of maintaining a list without duplicates.

For the breadth first search, a queue was used for its convenient enqueue and dequeue behavior. It also took advantage of vectors to ultimately store the shortest path in order to utilize the data structure's quick method to easily reverse a list. This was required due to the way in which predecessor nodes from the provided end node were obtained.

Implementation Difficulties

One implementation difficulty encountered involved generating random graphs for timing metrics. We could not rely on seeding a pseudo-random number generator with the current time when it was called thousands of times within a millisecond. At first, we overcame this by implementing the xorshift algorithm for pseudo-random number generation. More difficulties arose by attempting to ensure that the graph is connected, acyclic, and that a shortest path exists. Ultimately we generated graphs using the scenario that the graph is simply a line of nodes between the source and sink.

Theoretical and Empirical Analyses

Theoretically, the breadth first search algorithm runs in linear time with respect to the number of nodes and number of edges. Based on our implementation, each node is visited at least once. No other computations are involved other than storing the shortest path and minimum capacity, so it runs in

O(V+E). This is demonstrated empirically in figure 1. The Ford-Fulkerson implementation relies on the run time of the breadth first search to obtain the shortest path while one exists. Once it is obtained, it traverses over each node within the shortest path. Since the shortest path is based on the number of edges, this implementation executes in $O(VE^2)$ in the worst case. The empirical results are shown in figure 2, which correspond to a much better case and appears almost linear. The algorithm that solves the image segmentation problem relies on Ford-Fulkerson, though many optimizations were made to prevent it from running in quadratic time. The graph is created and the image is segmented in linear time O(E), and the Ford-Fulkerson algorithm is executed to verify no nodes were misplaced by the greedy approach, as seen in figure 3.

In order to accurately test the implementation for validity and to verify the predicted execution time, a series of test cases were created. For the timing tests, graphs were generated based on a specified number of vertices and nodes. The graphs were linearly connected in nature in order to guarantee that the graph was directed and acyclic. These were used for breadth first search and Ford-Fulkerson timing metrics. For the image segmentation timing metrics, forty-five PGM images were obtained of varying sizes. Ten test cases for the validity testing were created for breadth first search and Ford-Fulkerson, which included examples from the book. Each test case corresponded with a known shortest path, minimum capacity for the path, and maximum flow. The image segmentation validity testing was performed manually via visual observation.

Figure 1. Breadth First Search Timing

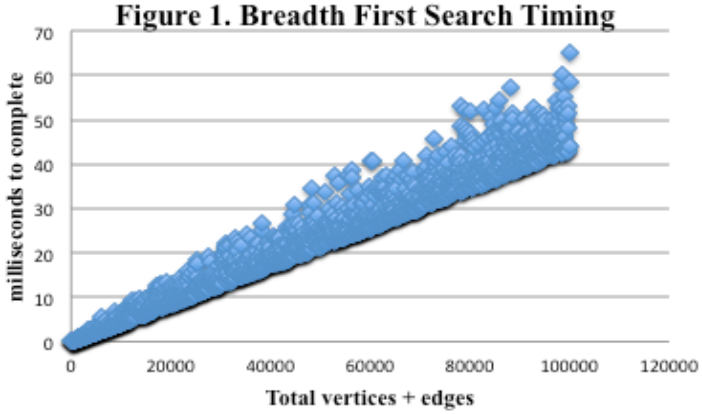


Figure 2. Ford Fulkerson Timing

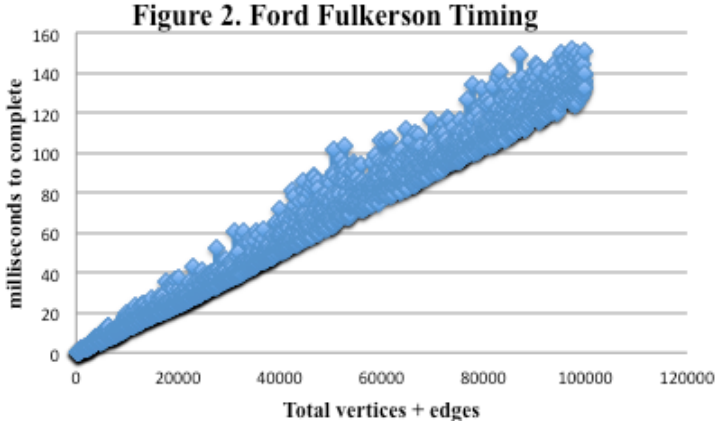
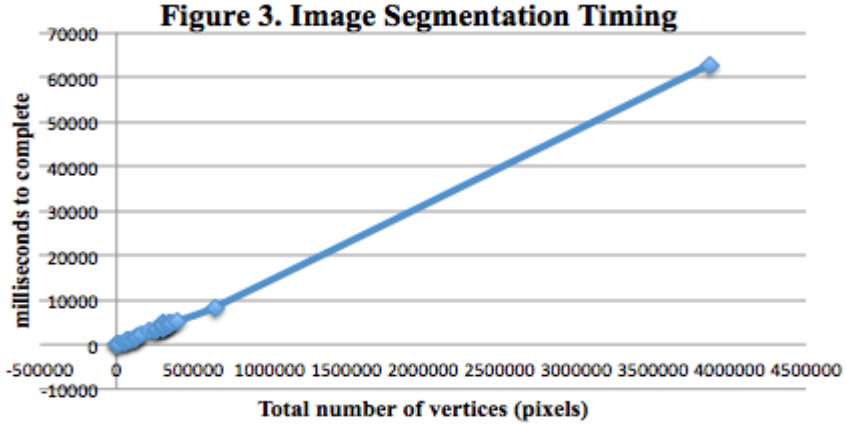


Figure 3. Image Segmentation Timing



Efficiency, Modularity, and Generalizability

The aforementioned algorithms were implemented with extensive optimizations. Specifically, the image segmentation solver runs much faster than the original implementation due to edge trimming during graph construction. Removing edges before executing the Ford-Fulkerson algorithm greatly reduces run time. In terms of modularity, each algorithm was self- contained to its own method within a

tools namespace. The image segmentation has also been generalized to such an extent that switching whether to preserve the foreground or the background is trivial. Graphs could also be extended to include negative numbers, but presently only integer node names are allowed. In addition, because a PGM contains a graph, the image segmentation could also be extended to run on another type of image.

With our approach, a few sacrifices had to be made. Most of these are in the form of space complexity. In the process of image segmentation we load the entire image into memory in a multidimensional array as it was the most efficient way to assemble the adjacency list graph for the segmentation, However, this was reused and provided us a very simple mechanism for updating the new output image as it was merely a matter of using the nodes identification numbers in the adjacency matrix in order to find the values in the multidimensional array, and then a simple iteration to output the result. Similarly, arrays were used to hold path neighbors and weights in the breadth first search. While extra space was used, it is utilized to ultimately reduce any unnecessary additional traversals such as walking a breadth first search tree or setting visited behavior inside the node objects.

Sample Results

The resulting image segmentation was quite successful. The foreground is separated from the background, and the background is overwritten with the maximum grayscale values. Sample results are shown in figure 4. The original input image is displayed on the left, and the processed output image is displayed on the right.

Figure 4. Image segmentation performed on the original PGM image (left) and its result (right).

Contributions

Drew was the team leader and coordinator of the project. He implemented the command line interface, the Ford-Fulkerson algorithm, and part of the image segmentation solution. He also developed the algorithm design, analysis, and powerful optimizations for breadth-first search, Ford-Fulkerson, and the image segmentation solver, and coordinated the team's effort in their comprehension of the algorithms that were required for the project. He also kept the program up to date with coding standards and ensured support for multiple operating systems. Drew also fixed various implementation errors within the program and was crucial in the debugging effort. He wrote the project's README, prepared the presentation slides, and contributed to the report documentation.

Bharath wrote part of Ford-Fulkerson code and wrote some test cases. He gathered these test cases from text book, some of them from other sources like wikipedia and some test cases are developed with his course work knowledge. He computed shortest path, maxflow, minimum capacity of the test cases. Bharah was involved in analysis of the image segmentation problem with the team mates. He helped in preparing presentation slides, writing in the report work and coding standards.

Heather implemented the support for creating graph objects in adjacency list format, the breadth first search algorithm, parsing a PGM file with its corresponding graph construction, and part of the image segmentation solution. She also implemented the graph generation algorithm for the timing metrics, and created the timing metrics for breadth first search and Ford-Fulkerson. She implemented the unit tests for breadth-first search and Ford-Fulkerson, which used the provided test cases and their calculated expected output. Heather executed an extended timing metric collection for a large data set, created the timing charts, predicted the time complexity, justified the data structures used, and collaborated with the team in preparing the report documentation and presentation.