Michael Crouse and Christopher Stoll

Dr. Duan

635 Advanced Algorithms

February 10, 2014

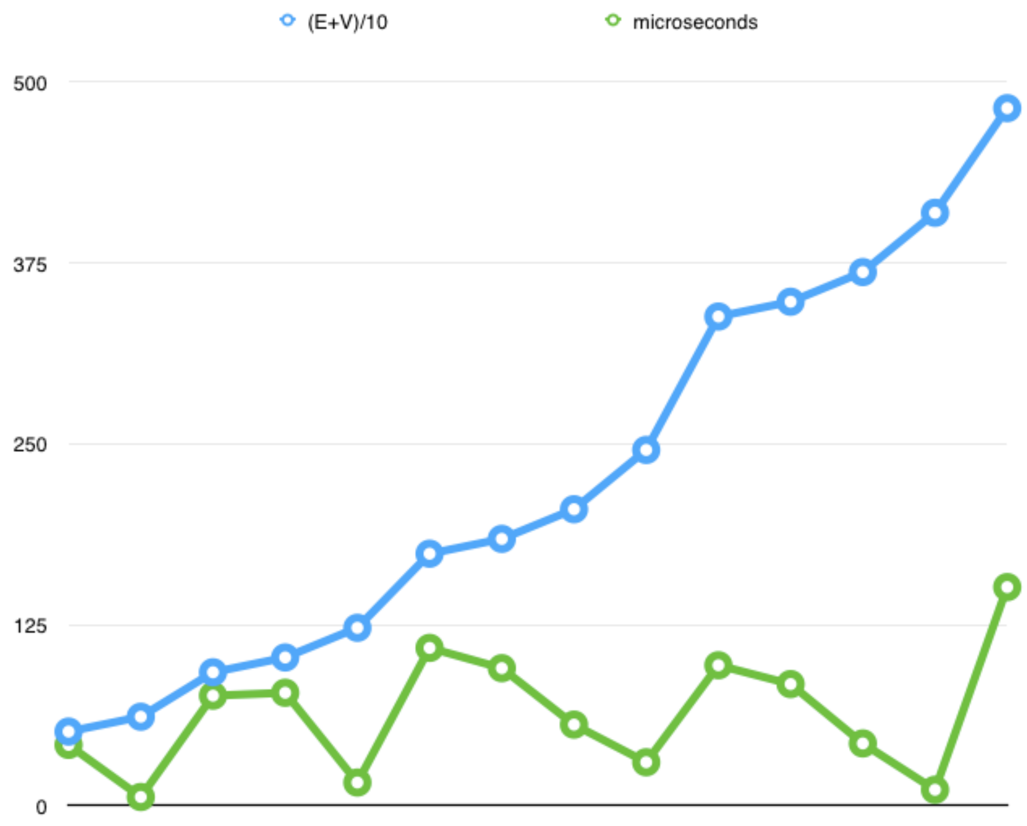
The objective of this project was to use the Ford-Fullkerson network flow algorithm to solve an image segmentation problem. The use of breadth first search (BFS) was required to find paths from the source node to the sink node in the Ford-Fullkerson algorithm. To apply a flow network to segmenting an image, each pixel in the image was treated as a vertex within a flow network. Each vertex was connected to adjacent vertices by edges forming a von Neumann neighborhood with a range of one. The capacity of the edges was determined by the difference in the pixels' z-dimensions. Since the images used in this project were grey-scale, the z-dimension was the 8-bit darkness of the pixel; a black pixel had a z value of 0 and a white pixel had a z value of 255.

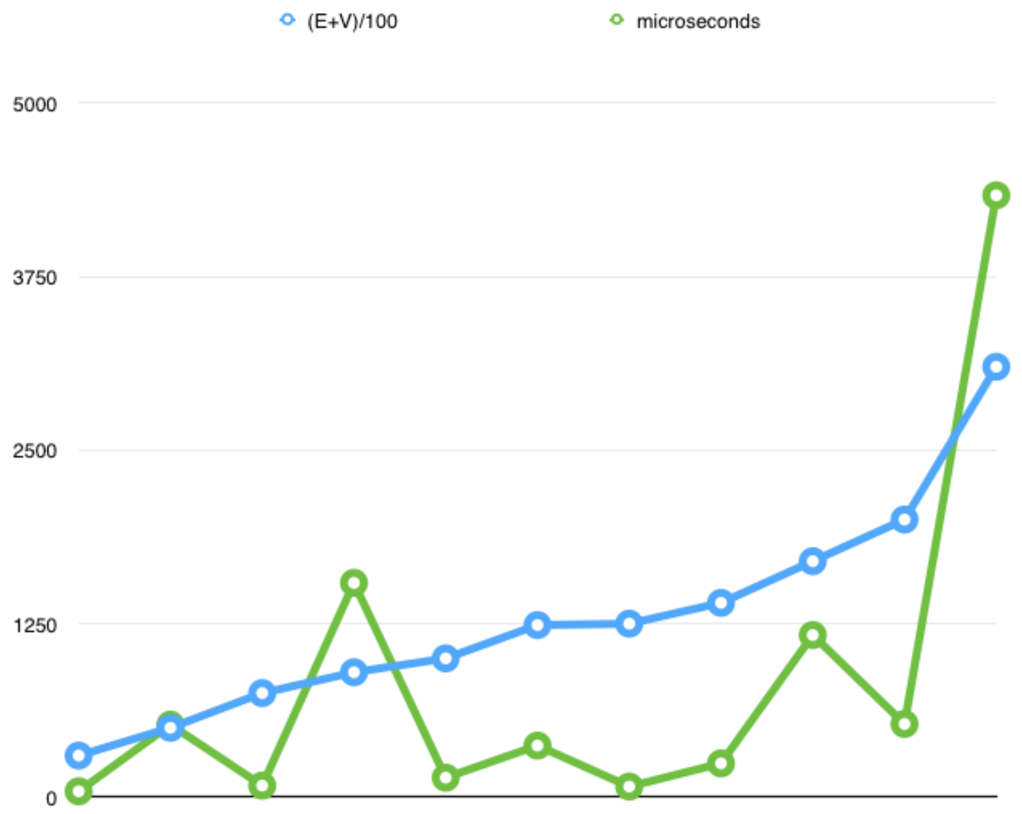
For the purposes of this project, the goal of image segmentation was to separate the background from the foreground of the greyscale image and either white-out or black-out the background of the image. In order to find the boundary between the foreground and the background we searched for the line of highest contrast, this is where the z-dimension differences are the highest; the z-dimension difference between a white pixel, with a value of 255, and a black pixel, with a value of 0, is 255. That line of maximum contrast should correspond to the the minimum cut. According to the max-flow min-cut theorem, the maximum flow from the source to the sink of a flow network corresponds with the minimum cut required to stop the flow from the source to the sink. So, after the Ford-Fulkerson algorithm is run we can berform another breadth first search on the residual graph from the source out to where the residual capacity is zero. This defines an area that should correspond to the background of the image.

We started by implementing breadth first search. We decided to use C, so we had to create node and graph structures as well as functions for handling queues and stacks. The graph was implemented using an adjacency matrix to reduce memory complexity even though this made the programming slightly more challenging.

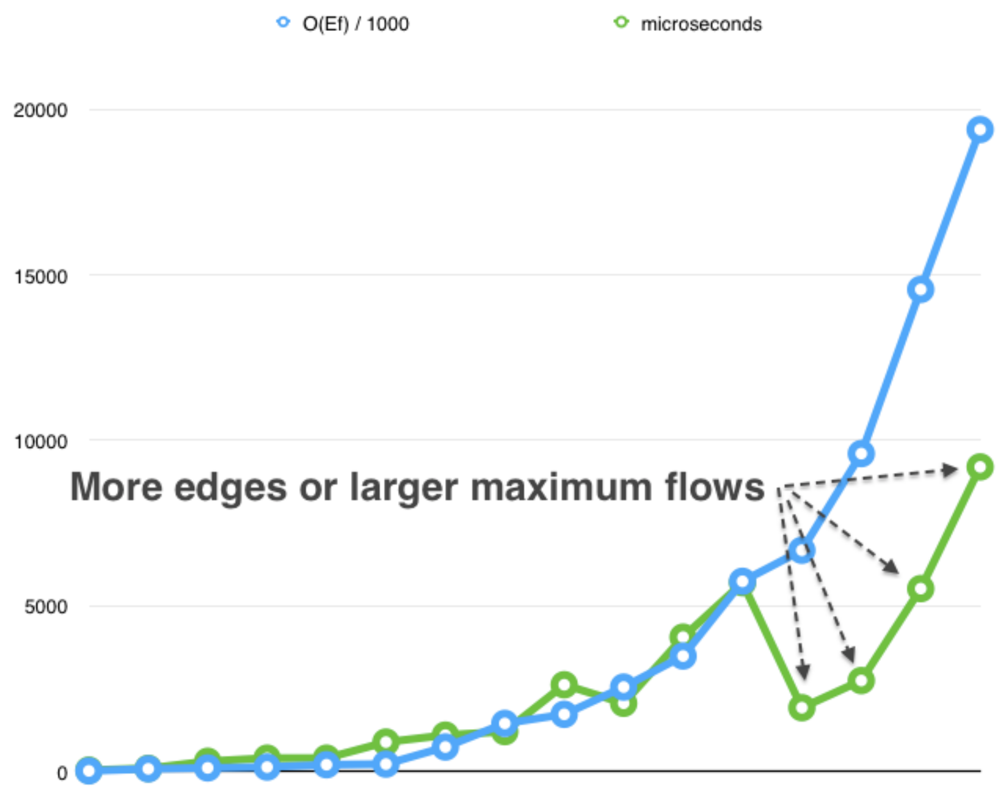
Once the breadth first search was implemented we ran some analysis to verify that it was performing as expected. We expected that in the worst case breadth first search should run in (|E| + |V|) time. The chart below show the results of our empirical analysis for a number of various sized graphs. Each graph was run 25 times, obvious outliers were excluded, then the times were averaged. The (|E| + |V|) values were scaled down so that the initial points roughly correspond; this is to make the asymptotes easier to compare. The time taken to run each graph does not form a smooth asymptote, this is due to differences in the graph densities. We used different densities to check a wider range of operating parameters.

It is worth noting that, even with very large graphs, our unoptimized C implementation of the breadth first search, as well as Ford-Fulkerson, ran so quickly that we had to measure run-times in microseconds (1/1,000,000 second). On these small time scales it seems that the impact of operating system optimizations, such as Timer Coalescing on OS X Mavericks, have a large impact upon run-times. We ran our analysis program multiple times, and occasionally a graph would report taking orders of magnitudes longer to run. We used averages and outlier exclusion to counter this, but the operating system overhead constants seem to have an outsized impact at this time scale.


(E+V)/10
microseconds
500
375
250
125
0


(E+V)/100
microseconds
5000
3750
2500
1250
0

Once the breath first search was working as expected we implemented the Ford-Fulkerson algorithm. Analysis shows that our Ford-Fulkerson performs in O(Ef), the number of edges times the maximum flow, as expected. Perhaps due to the operating system overhead constants mentioned above, the algorithm performs relatively better on larger graphs or graphs with larger maximum flows.



Finally, after Ford-Fulkerson was complete, we were able to implement the image segmentation portion of the program. We started by adding in code which can read and write PGM files. Since the files represented an image we used a matrix to store the points. We then created a graph with all the associated edges. The difference in z-value was used as the capacity of the edge. In order to select our source and sink we original-ly added an external source connected to all the nodes on the left and right sides of the image. We were not getting the results we expected, so we changed the program to search the image matrix for the darkest and lightest regions, then attached the source node to the dark vertices and the sink node to the lightest vertices. The idea was to search out from the dark regions where flow should be the highest, stopping at the minimum cut. The effect of this approach, however, was that the dark regions were cut out of the image instead of the light regions. Despite the dark regions being cut out instead of the light regions, the program is functioning as required; the results are just inverted.

For future projects in which we are unfamiliar with the algorithms involved, we believe that it may be easier to use a higher level language than C, at least until all the implementation steps are well understood.