# Graph-cut based segmentation of images

ANSHUMAN SURI, 2014021, Indraprastha Institute of Information Technology
HARSHVARDHAN KALRA, 2014043, Indraprastha Institute of Information Technology

**ACM Reference format:**
Anshuman Suri, 2014021 and Harshvardhan Kalra, 2014043. 2016. Graph-cut based segmentation of images. 1, 1, Article 1 (January 2016), 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 PROBLEM STATEMENT

Given an image as input, the problem is to segment it into two classes, namely the background and the foreground. Several serial implementations exist for this, but are painstakingly slow since they are executed on the CPU, which is inherently serial. Some of these algorithms reduce the given image into a graph, on which certain algorithms are run to get a desired segmentation. These algorithms have a significant ratio of parallel component, which means if it can be ported to run on a GPU, it is possible to achieve significant speedups. We aim to implement some well known algorithms to achieve image segmentation on a GPU (and if it can run in real-time, extend it to run on videos).

## 2 LITERATURE AND ALGORITHM DETAILS

We read the papers by Boykov and Jolly[1] and a CUDA implementation of the push-relabel algorithm done by Vineet and Narayanan[2], as well as referring to two similar papers on implementing graph-cut algorithms on CUDA. We intend to use these to benchmark our code 's performance.
The task of performing image segmentation can be reduced to finding the minimum cut in a graph, where each node is represented by a pixel (or a group of pixels, i.e., voxel), and the edge weights are defined according to the required type of segmentation. Most image segmentation techniques model a given image (or volume) into a graph, and then play around with the edge weights to produce a graph which can be used to find an appropriate segmentation. Thus, once the image volume has been decomposed to a graph, we can use parallel algorithms for finding the minimum cut in such a graph to obtain a background-foreground segmentation.
Out of all the min-cut finding algorithms that can be executed in parallel to run on multiple processors, the push-relabel algorithm can be used to find such a cut, and seems to have the best performance so far.
The push-relabel algorithm for computing the min-cut works as follows:

- There are 2 procedures, namely push and relabel, which are responsible for producing the minimum cut. Each node has two associated information : a label which, apart from the source and sink, underestimates (gives a lower bound on) the distance of the node (in terms of the number of vertices) to the sink. Also, the excess flow of a node is defined as the amount of flow it receives minus the flow it sends out.


DOI: 10.1145/nnnnnnn.nnnnnnn

- The push sub-procedure does the following : for any node u, it checks the excess e(u) and u's label, and selects one of its neighbours v, such that the capacity of (u, v) is not saturated, and v's label is k - 1 (one closer to the sink). If there is such a node, u's excess is pushed through (u, v). This results in the excesses of u and v being updated, and either (u, v) being saturated, or u's excess becoming 0. Intuitively, this procedure tries to push flow one step closer to the sink.
- The relabel sub-procedure does the following : for any node u, the relabel procedure checks u's excess and label, and tries to push it to any lower-labelled neighbour of u. If all neighbours are having higher labels, the sub-procedure relabels u by providing it a minimum label so that a push might be possible. Initially, each node's excess is set to 0, other than those nodes having connections from the source. Irrespective of the order in which the push and relabel operations are performed, the algorithm is guaranteed to converge, and the time complexity is $O(EV^2)$ (when run serially).

We plan on using this algorithm to segment a given image in parallel. As of now, we also plan on trying another approach (that uses the BK algorithm) for segmenting images (which also involves reducing the image into a graph). If any of these two algorithms run in real-time for a given image, we also plan on extending it to run for a video sequence in real time (which can be further optimized, given the high correlation between any two consecutive frames).

## 3 IDENTIFIED TASKS, TIME-LINE AND WORK DIVISION

### 3.1 Identified Tasks

The intermediate tasks that we could think would of now are:

- Intermediate task 1 (Starting Feb'27) : Implement the push-relabel based algorithm for segmenting the given image. Include the option for the user to provide hard constraints while asking for the image to be segmented (for better results).
- Intermediate task 2 (Starting March'20) : Perform optimizations in the above code-base, and make it faster by using the approaches taught in class. Start implementing BK algorithm / extend above algorithm for video sequences. See if it can be run in real time, and benchmark performance.
- Final task 3 (Starting April'3) : Combine all of the above and come up with a final, optimized code-base that can efficiently and quickly segment images (and reach performance close to what's mentioned in the papers that we've read).

### 3.2 Work Division

- Implementing image segmentation on CPU (for benchmarking) : Anshuman
- Implement push-relabel subroutine on GPU : Harshvardhan
- Combine push-relabel with edge-weight calculation and come up with final image segmentation code: Anshuman
- Perform optimizations on above code: both
- Extend above code to video sequences/ implement BK on CPU+GPU : Harshvardhan/Anshuman
- Optimize above code: Both
- Final report + Summarizing: Both

## REFERENCES

[1] Boykov, Yuri Y., and M-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. Computer Vision, 2001
[2] Vineet, Vibhav, and P. J. Narayanan. CUDA cuts: Fast graph cuts on the GPU. Computer Vision and Pattern Recognition Workshops, 2008.
[3] Hussein M, Varshney A, Davis LS. On implementing graph cuts on CUDA. First Workshop on General Purpose Processing on Graphics Processing Units, 2007.

[4] P. Harish and P. J. Narayanan. Accelerating large graph algorithms on the GPU using CUDA. International Conference for High Performance Computing, 2007.