Graph Cuts for Binary Segmentation of Image

Chitesh Tewani[1] and Srivatsan Iyer[2]

***Abstract*— This report presents a detailed look at our implementation of binary segmentation of images using Graph Cuts for a Network Flow problem. It also presents a alternative approach that improves the performance.**

I. Introduction

Binary segmentation of images, in the domain of Computer Vision, is the process of partitioning the image such that each area belonging to a partition represent a unique object in the scene (using its boundary). This project segments an image and separating the foreground from the background.

The field of Computer Vision and Computer vision technology has captured a lot of attention in research and industry during the past few years. The Computer Vision technology have applications in domain of consumer market, automotive, robotics, medical, security, etc.

Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. The task of image classification and identification rely on segmentation. This field also involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding.

We have implemented the procedure of binary segmentation of images which uses the concept of network flow algorithms. The original problem can be transformed into a graph-cut which can be solved by computing the maximum flow on the flow network, generally Max-flow min-cut.

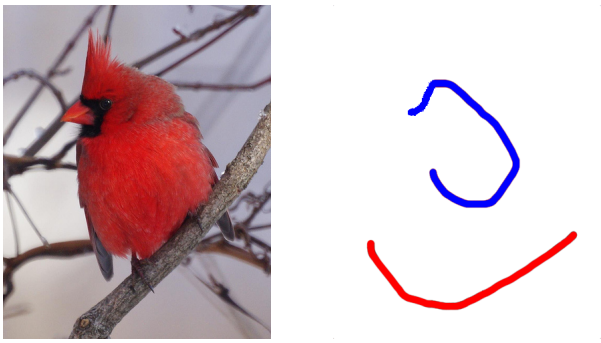
II. Related Work

Greig et al. [5] were first to discover that powerful min-cut/max-flow algorithms from combinatorial optimization can be used to minimize certain important energy functions in vision. They consider energy in the context of maximum of posteriori estimation of Markov Random Fields (MAP-MRF). The energy equation can be represented as:

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{(p,q) \in N} V_{p,q}(L_p, L_q)$$

where $L = \{L_p | p \in P\}$ is the labeling of the image P, $D_p(.)$ is the data penalty function, $V_{p,q}$ is an interaction potential, and N is a set of all pairs of neighboring pixels. Typically, the data penalties $D_p(.)$ indicate individual label-preferences of pixels based on observed intensities and pre-specified likelihood function. Interaction potentials $V_{p,q}$ encourage spatial coherence by penalizing discontinuities between neighboring pixels.

[1]Chitesh Tewani (ctewani@indiana.edu)
[2]Srivatsan Iyer (srriyer@indiana.edu)
[3]Original image from Rob Crandall's Flickr account and seed image provided by Dr. David Crandall in CSCI-B657



(a) Original Image (b) User hint

Fig. 1: Original image and user hint[3].

Early attempts to use combinatorial graph cut algorithms in vision were restricted to image clustering [8]. In the late 90s a large number of new computer vision techniques appeared that figured how to use min-cut/max-flow algorithms on graphs for solving more interesting non-binary problems.

Several recent papers studied theoretical properties of graph constructions used in vision. The question of what energy functions can be minimized via graph cuts was addressed in [9]. The geometric properties of segments produced by graph-cut methods were investigated in [10]. This work studied cut metric on regular grid-graphs and showed that discrete topology of graph- cuts can approximate any continuous Riemannian metric space. The results in [9] established a link between two standard energy minimization approaches frequently used in vision: combinatorial graph-cut methods and geometric methods based on level-sets.

In a recent paper by Boykov [1], they experimentally compared the running time of several min- cut/max-flow algorithms on graphs typical for applications in vision. The paper implemented consider both Goldberg-Tarjan style push-relabel algorithms [11], methods based on augmenting paths using Ford-Fulkerson [3] and Dinic algorithm [12]

III. Background

The implementation for binary segmentation of images is similar to work by Boykov [1]. We consider an image as a directed weighted (capacitated) graph $\mathcal{G} = <\mathcal{V}, \mathcal{E}>$ consists of a set of nodes $\mathcal{V}$ and a set of directed edges $\mathcal{E}$ that connect them. For an image of width w and height h contains $\mathcal{V} = w \times h$ nodes or pixels and contains number of edges $\mathcal{E} = (w-1) \times h + w \times (h-1)$ to be precise.

All the pixels or nodes have neighboring 4-connected pixels, which share an edge. In addition to these nodes and edges, we have two special nodes that we add to the graph *foreground node* (N_f) and *background node* (N_b). These two

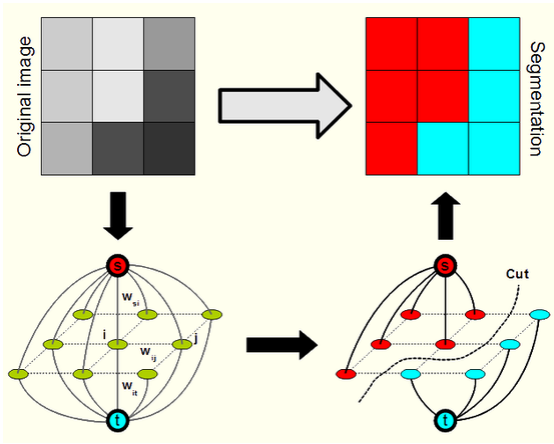


Fig. 2: Graph Structure[13].

nodes are connected to all the $w \times h$ pixels. Throughout the paper, we will refer to these nodes as $N_{i,j}$ where i is the row number, and j is the column number. The structure of the graph so formed is illustrated in Fig. 2. With appropriate weights on each of the edges, a graph cut that disconnects N_f from the N_b will yield the required segmentation. The nodes that remain connected to N_f will be labeled as foreground, and the nodes that remain connected to N_b will be labeled as background.

To represent the symbols, we denote the weight of the edge between a node and its 4 connected neighbors as $W_{x,y}$ where x and y can stand for any of the nodes $N_{i,j}$, N_f or N_b. Assuming a function $I(n)$ that accepts a node and returns a 3-tuple representing the red, blue and green channels, RGB Color mode of the original image. Let us further define $E(x, y)$ to be euclidean distance between x and y where both x and y are typically the 3-tuples returned by the function I. Weight of the edge between two adjacent nodes needs to higher when they have similar intensities and vice versa. As the maximum representation in the RGB color of a channel is 255 and the minimum is 0, we scale between the Euclidean distances of the color representation. In our implementation we have defined the weights as:

$$W_{N_{a,b},N_{x,y}} = 1 - \frac{E(N_{a,b}, N_{x,y})}{E((0,0,0),(255,255,255))}$$

To calculate the weights of each of the pixel nodes with both N_f and N_b, we need to use the hints provided by the user. Let us denote $S_f(i, j)$ to be a boolean valued function that returns a 1 if the pixel (i, j) is marked as a foreground pixel in the seed image, or zero otherwise. Similarly, let us denote $S_b(i, j)$ to return a 1 or a zero depending on whether the pixel (i, j) is marked as a background pixel in the seed image.

We calculate the mean and variance of all the pixels of the image that fall under the location of foreground and background marked pixels in the seed image (*"user hint"*). Let us use μ_f and μ_b to represent the mean of all the red, blue, green intensities of foreground and background pixels respectively. Similarly, let us denote variances in these pixels using σ_f and σ_b. We now, can create a multivariate Gaussian distribution G with mean as μ_f and covariance matrix σ_f. The weight of a node $N_{i,j}$ and N_f is:

$$W_{N_{i,j},N_f} = \begin{cases} \infty, & \text{if } S_f(i,j) = 1 \\ 1 - \frac{G(\sigma_f) - G(I(i,j))}{G(\sigma_f)}, & \text{otherwise.} \end{cases} \quad (1)$$

Similarly, weight for a node connected to N_b is:

$$W_{N_{i,j},N_f} = \begin{cases} \infty, & \text{if } S_b(i,j) = 1 \\ \frac{G(\sigma_f) - G(I(i,j))}{G(\sigma_f)}, & \text{otherwise.} \end{cases} \quad (2)$$

The above equations are used to calculate the weights of various edges between nodes in the graph.

IV. Implementation Details

The overall implementation includes implementation[1] of the core data structures, reading and parsing images into graphs, calculating weights, implementing Edmonds-Karp algorithm and various improvements on it. This report[2] focuses on the representation of Network Graph, Edmonds-Karp algorithm and a randomized improvement of the algorithm.

A. Network Graph

The network flow graph representation of an image is Fig. 2. The image of width w and height h is represented by matrix of pixels in $w \times h$. Each of the pixels are connected to the 4 neighboring pixels. The *foreground node foreground node* (N_f) and *background node* (N_b) are connected to all the $w \times h$ pixels.

We are using a adjacency-list based representation of the network flow graph. The object-oriented representation of the graph $\mathcal{G} = <\mathcal{V}, \mathcal{E}>$ contains:

1) Node -
   - Node object representing a pixel consists of a list of of Edges, connecting to the neighboring nodes.
   - Node contains the position or 2-D co-ordinates in the image
   - Node also contains the intensity in RGB Color mode. This is used to calculate the weight between two pixels or neighboring nodes.
   - Node consists of a *label*, which can be foreground, background or unknown, based on the labels from user-hinted image
   - Foreground (N_f) node and background node (N_b) are special nodes with no intensity and placeholder position coordinates
2) Edge -
   - An Edge contains 2 node objects forming the edge
   - Edge object represents the weight between the neighboring pixel nodes. This weight given by

[1]Source code - http://github.iu.edu/srriyer/ImageSegmentationGraphCuts

[2]Srivatsan and I both worked on implementation of the core Edmonds-Karp algorithm. I continued to work on designing and implementing the network graph structure, also the randomized implementation of BFS

$W_{N_{a,b},N_{x,y}}$ in III, which is the capacity of flow through the edge

- The Edge object connecting the pixel with the *foreground node* (N_f) is given by 1 and the pixel connecting the *background node* (N_b) is given by 2 which is the capacity of flow through the edge
- Edge additionally contains a residual capacity equal to the capacity which aids in calculation for Max-Flow Min-Cut algorithms

The time complexity of construction of the graph in form of adjacency-list is $\mathcal{O}(\mathcal{V})$ and space complexity of $\mathcal{O}(\mathcal{V}+\mathcal{E})$, where $\mathcal{V}$ is the number of vertices and $\mathcal{E}$ is the number of edges.

B. Edmonds-Karp Algorithm

Once we have a graph represented in terms of nodes and edges, we can apply EdmondsKarp[3] algorithm. The algorithm tries to find augmenting path until there is no more paths with any available capacity. The saturated edges represent cuts and a simple graph search for all reachable nodes (pixel nodes) from the source node N_f will give us all the foreground pixels. All other unreachable pixel will be labeled background pixel. The algorithm being used is described 1 below.

The time-complexity of Edmonds-Karp can be given as $\mathcal{O}(\mathcal{V}\mathcal{E}^2)$, where $\mathcal{V}$ is the number of nodes and $\mathcal{E}$ is the edges

Algorithm 1 Edmonds Karp Min Cut

```
function MIN-CUT(all-nodes, source, target)
    while True do
        path ← FIND-MIN-PATH(source, target)
        if not exists(path) then
            break
        min-edge ← MIN(path)
        for p in path do
            p.residual ← p.residual − min-edge.capacity
    fg-nodes ← REACHABLE-BFS(source)
    bg-nodes ← all-nodes \ fg-nodes
    return fg-nodes, bg-nodes
function REACHABLE-BFS(source)
    visited ← set()
    queue ← [source]
    while not EMPTY(queue) do
        front ← POP-QUEUE(queue)
        if node in visited then
            break
        SET-ADD(visited, node)
        for edge in node.edges do
            if edge.residual ≥ threshold then
                n ← GET-OTHER-NODE(edge, node)
                APPEND-QUEUE(queue, n)
    return visited
```

[3]Edmonds-Karp algorithm is a specialization of Ford-Fulkerson's algorithm where the augmenting path is found using BFS.

C. Randomized Edmonds-Karp Algorithm

The implementation of randomized Edmonds-Karp algorithm is a randomized-version of the algorithm which finds the minimum path from *foreground node* to *background node*. The algorithm used to find the minimum path is Breadth First Search, as it returns the shorted path.

In randomized version of breadth first search, we shuffle the edges to be explored or visited in the BFS, which randomizes the path it could start exploring from.

The time-complexity of shuffling is typically, $\mathcal{O}(n)$, where n is the number of elements in the list. In this case, the number of edges explored or the branching factor is a constant, 4-neighboring nodes. Hence, the overall complexity of the randomized algorithm is the same as Edmonds-Karp Algorithm, that is, $\mathcal{O}(\mathcal{V}\mathcal{E}^2)$.

V. Experimental Results

(a) Original Image (b) Foreground (c) Background

Fig. 3: Segmentation using Graph Cuts.

A. Running Time

B. Computing Environment

The experiments were run on an Core i7 HQ 4^{th} gen CPU with 16 GB of RAM in an Arch Linux environment. The entire code-base is written in Python and compatible with Python 2.7+.

C. Input

The input requires an input image and an user-hinted image. The user-hinted is a seeded image of the same resolution and size as the original image. It also marks few pixels representing the background and foreground. A input image and an user-hinted image is represented in Fig. 2

The experiments were performed on original image of 400×500 pixel size. We scaled the image down to various sizes to investigate the effect of different input sizes on the running time of the algorithms.

D. Output

The results of the experiment from min-cut algorithms will generate two separate images, representing the foreground and background. The pixels in the same set as the source, i.e. foreground pixels, form the foreground image. The pixels in the same set as the sink, i.e. background pixels, form the background image.

The background image has foreground pixels suppressed (made black) and foreground image has background pixels

suppressed (made black). The output is represented in Fig. 3

We have performed experiments for binary segmentation of image on the following algorithms:

1) Edmonds-Karp Algorithm
2) Edmonds-Karp Algorithm using randomized BFS
3) Edmonds-Karp Algorithm using bi-directional BFS
4) Edmonds-Karp Algorithm using randomized bi-directional BFS

TABLE I: Run time comparison of Edmonds-Karp (in secs)

$w \times h$	BFS	Rand. BFS	Bidirect.	Rand. Bidirect.
40×50	11.72	10.31	4.91	4.68
48×60	20.57	19.11	7.66	5.74
56×70	28.29	28.48	9.90	11.83
64×80	45.31	53.78	14.66	19.72
72×70	70.34	71.92	30.83	25.40
80×100	87.48	102.53	27.51	30.41
88×110	156.33	154.65	47.53	72.70

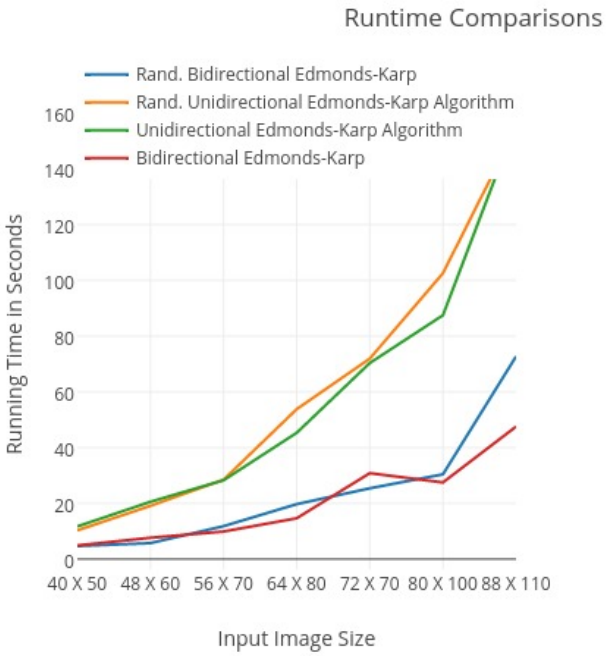


Fig. 4: Run time comparison

Table I illustrates the running times of the four algorithms for different input sizes[4] by scaling down the input image. The results of our experiments, we give the running times in seconds.

[4]Parameters were tuned to attain this segmentation. Same parameters were not used for complexity analysis because the focus was not on more accuracy but on running time.

The run-time comparison represented in Fig 4 (generated in Plotly[5]) demonstrates that bidirectional improvement on Edmonds-Karp algorithm performed significantly better compared to Breadth First Search implementation Edmonds-Karp algorithm. However, the randomized version of both the algorithm performed relatively the same to their respective algorithms.

VI. Conclusion

Through these experiments we demonstrate that Images, with some user hint, can be represented as a graph. This graph upon a min-cut will yield two sub disconnected sub-graphs thereby segmenting the image. The performance of the segmentation depends a lot on the path finding algorithm chosen.

We compared breadth first search implementation of Edmonds-Karp algorithm and improved Edmonds-Karp algorithm by using bi-directional breadth first search. Along with it, we experimented with the randomized search strategy. We could observe that non-randomized bi-directional search strategy is much more efficient compared to Edmonds-Karp Algorithm.

References

[1] Y. Boykov and V. Kolmogorov, An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision, IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI), vol 26, no.9, pp 1124-1137, Sept 2004.

[2] Y. Boykov, O. Veksler and R. Zabih, Faster approximate en- ergy minimization via graph cuts, IEEE transactions on Pattern Analysis and Machine Intelligence (PAMI), vol 23, no. 11, pp 1222-1239, Nov 2001.

[3] L. Ford and D. Fulkerson, Flow in Networks, Princeton University Press, 1962.

[4] A.V Goldberg and R. E. Tarjan, A new approach to the maximum-flow problem, Journal of the Association for Computing Machinery, vol 35, no. 4, pp 921-940, Oct 1988.

[5] D. Greig, B. Porteous, and A. Seheult. "Exact maximum a posteriori estimation for binary images", Journal of the Royal Statistical Society, Series B, 51(2):271279, 1989.

[6] T.H. Cormen, C.E. Leiserson and R.L. Rivest, Introduction to Algorithms, McGraw-Hill, 1990.

[7] Y. Boykov, O. Veksler, and R. Zabih. "Markov random fields with efficient approximations". In IEEE Conference on Computer Vision and Pattern Recognition, pages 648655, 1998.

[8] Zhenyu Wu and Richard Leahy. "An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(11):11011113, November 1993

[9] Vladimir Kolmogorov and Ramin Zabih. "What energy functions can be minimized via graph cuts", IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(2):147 159, February 2004.

[10] Y. Boykov and V. Kolmogorov. "Computing geodesics and minimal surfaces via graph cuts", International Conference on Computer Vision, volume I, pages 2633, 2003.

[11] Andrew V. Goldberg and Robert E. Tarjan. "A new approach to the maximum-flow problem", Journal of the Association for Computing Machinery, 35(4):921940, October 1988.

[12] E. A. Dinic. "Algorithm for solution of a problem of maximum flow in networks with power estimation", Soviet Math. Dokl., 11:12771280, 1970.

[13] Graph Structure *http://mygsoc.blogspot.com/2013/07/graph-cuts.html*

[14] Maximum flow augmenting path *https://www.topcoder.com/community/data-science/data-science-tutorials/maximum-flow-augmenting-path-algorithms-comparison/*

[5]Plotly - www.plotly.com