A Gradient Descent Approximation For Graph Cuts

Alparslan Yildiz and Yusuf Sinan Akgul

Computer Vision Lab., http://vision.gyte.edu.tr
Department of Computer Engineering, Gebze Institute of Technology
Gebze, Kocaeli 41400 Turkey
{yildiz, akgul}@bilmuh.gyte.edu.tr

Abstract. Graph cuts have become very popular in many areas of computer vision including segmentation, energy minimization, and 3D reconstruction. Their ability to find optimal results efficiently and the convenience of usage are some of the factors of this popularity. However, there are a few issues with graph cuts, such as inherent sequential nature of popular algorithms and the memory bloat in large scale problems. In this paper, we introduce a novel method for the approximation of the graph cut optimization by posing the problem as a gradient descent formulation. The advantages of our method is the ability to work efficiently on large problems and the possibility of convenient implementation on parallel architectures such as inexpensive Graphics Processing Units (GPUs). We have implemented the proposed method on the Nvidia 8800GTS GPU. The classical segmentation experiments on static images and video data showed the effectiveness of our method.

1 Introduction

Graph cuts have been extensively used in computer vision in solving a wide range of problems such as stereo correspondence, multi view reconstruction, image segmentation, and image restoration. One of the reasons of their popularity is the availability of practical polynomial time algorithms such as [1] and [2]. As a result, the literature includes many successful graph cut based computer vision systems.

Minimum cut on a graph can be formulated as finding the maximum flow that can be pushed from the source node to the sink node on the graph through the links between the nodes with known capacities. Ford and Fulkerson [9] showed that finding the minimum cut on a graph that divides the nodes into two distinct sets as source and sink is equivalent to finding the maximum flow from source node to the sink node in the same graph. In computer vision, the algorithm of Boykov and Kolmogorov [1] is the most popular one. It computes the maximum flow using a modified version of the Ford-Fulkerson maximum flow algorithm. The algorithm finds augmenting paths to push flow using two search trees, one rooted at the source node and the other rooted at the sink node. Once these two search trees meet, an augmenting path from source to sink is found and the maximum possible flow is pushed through this path. As an improvement to standard augmenting path algorithm, this algorithm searches for the next augmenting path using the same search trees, which significantly improves the total running times. Although this algorithm is successfully used for medium sized image segmentation and small sized 3D segmentation, it is not practical for high resolution images, 3D data, and realtime dynamic segmentation applications due to large amounts of memory requirements and slower running times.

The literature includes methods to speed-up the graph cut computation for larger graphs, such as [3] which proposes a coarse to fine scenario. A speed-up for realtime purposes is suggested by [4] which uses an initial flow to compute the solution. Another speed-up is proposed by [5] which uses the residual graph from previous frames for the graph cut computation of the next frames of a video sequence. Another direction for graph cut speed-up employs Graphical Processing Units (GPUs). Dixit et. al [6] implemented the push relabel maximum flow algorithm on the GPU. Vineet and Narayanan [7] showed that the recent GPUs, such as the Nvidia GTX280 [10], outperform the best sequential maximum flow algorithms by a factor of around 10 in running times.

Recently, Bhusnurmath and Taylor [8] have formulated the graph cut problem as an unconstrained l_1-norm minimization and solved it using the Newton's method on the GPU. A very interesting property of their work is that, different from the other methods above, it is based on solving the graph cut problem using the minimum cut formulation rather than the maximum flow formulation. The minimum cut formulation directly solves the labels of the nodes and as they are known to converge to one of the labels, rounding errors are less of a problem. Another very useful property of the minimum cut formulation is that, as it uses the labels of the nodes directly, the initial estimate of the solution can be given without any processing on the graph, which is not possible with the maximum flow formulations. In spite of the novelty of [8], the method uses too much memory and it becomes impractical for some applications that have to deal with large graphs.

In this paper, we present a gradient descent approximation of graph cuts. Our method is similar to the minimum cut formulation given by Bhusnurmath and Taylor [8] but, we have several major advantages. Although the gradient descent method will require more iterations than Newton's method, it has some very important advantages over the Newton's method. The most important advantage of the gradient descent method is the dramatically lower memory requirements. For applications where the data structures for the Newton's method will not fit in the memory, our method can successfully work. Other advantages of our gradient descent method include the local computation of gradients, high parallelization and the ease of implementation. Finally, using the red-black order and the SOR (Successive Over-relaxation) method to speed-up the process even further can be considered as additional benefits of our method.

The reminder of the paper is organized as follows; we give an overview of the graph cuts and the minimum cut formulation in Section 2. The details of applying gradient descent method on the minimum cut formulation is explained in Section 3. We provide experimental results and running times of our method in Section 4 and finally we give concluding remarks in Section 5.

2 Graph Cuts

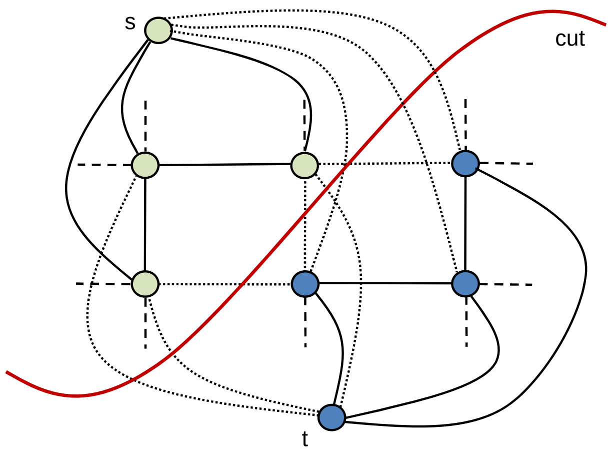


Fig. 1. Portion of a typical graph used in computer vision. A cut is shown partitioning the nodes to source and sink sets. Saturated edges that are forming the cut are indicated as dotted.

In this section we give an overview of the graph cuts and the minimum cut formulation as an unconstrained l_1-norm minimization. A graph $G(\mathcal{V}, \mathcal{E})$ is a set of n nodes $\mathcal{V}$ and m weighted edges (capacities) $\mathcal{E}$ that connect these nodes. There are two special nodes in the binary graph cuts formulation, the source node s and the sink node t. The special nodes s and t have edges to all other nodes, some of which may have 0 weights. The edges between s and other nodes, excluding the node t, are called source links and the edges between t and other nodes, excluding the node s, are called sink links. The source and sink links are also included in the set $\mathcal{E}$.

The maximum flow problem is to push as much flow as possible from the source node s to the sink node t. The weights of the edges between the nodes define the capacities that limit how much flow can be pushed through these edges. Thus, if there are no saturated edges, it means that there is still some flow that can be pushed from s to t. Hence, the maximum flow will saturate some of these edges. The max-flow/min-cut theorem [9] states that the minimum cut on a graph will include saturated edges and will cut the graph in two partitions such that some of the nodes will be in the source set and the other nodes will be in the sink set. The value of the minimum cut is the sum of the weights of the saturated edges in the cut and is equal to the value of the maximum flow. See Fig. 2 for a simple graph with saturated edges and a cut.

Since maximum flow and minimum cut is equivalent, solving either of these formulations will also reveal the solution of the other. In computer vision, binary labeling problems are usually defined in terms of minimum cut that will assign appropriate labels to pixels (nodes). However, the problem is usually solved via the maximum flow formulation. The maximum flow algorithm of the Boykov and Kolmogorov [1] is the most popular one in computer vision. Their algorithm implements a tuned version of the well-known Ford-Fulkerson maximum flow algorithm which finds augmenting paths between s and t and pushes flow through these augmenting paths until there is no augmenting path left.

Recently, Bhusnurmath and Taylor [8] has formulated the maximum flow problem as a linear programming problem and solved the minimum cut formulation as the dual of the maximum flow formulation. They give the minimum cut formulation as the minimization of

$$F(\mathbf{v}) = \sum_{i}^{m} w_i |(\mathrm{A}^T \mathbf{v} - \mathbf{c})_i|, \tag{1}$$

where w_i is the i-th edge capacity and m is the number of edges including the source and the sink links. The $n \times m$ matrix A represents the graph structure with n being the number of nodes. $\mathbf{c}$ is the vector of length m which indicates the source links with 1 and any other link with 0. The values of the nodes are represented by the vector $\mathbf{v}$ of length n. They proved that this unconstrained l_1-norm minimization will lead the node values $\mathbf{v}$ to either 1 (source) or 0 (sink).

The matrix A can get very large even for reasonable graphs and applying Newton's method to this formulation requires some careful work. The next section introduces our novel solution to this problem by using the gradient descent method to this formulation without building the matrix A, which lets the method work easily even for very large graphs.

3 Gradient Descent for Graph Cuts

Our gradient descent method for solving graph cuts is motivated by the increasing popularity of GPUs for general purpose computing in vision. As the GPUs have high number of parallel processors and many times more computational power than the CPUs, it is logical to move some intensive computations to the GPUs. Graph cuts are very popular for low-level vision and there are a number of GPU implementations for graph cuts as mentioned before. Our method for computing graph cuts is the application of well-known gradient descent minimization to the unconstrained l_1-norm formulation of the minimum cut.

The gradient descent algorithm to minimize a function $F(x)$ of Eq. 1 is given in Algorithm 1.

For the gradient descent method to work, $F(x)$ needs to be a smooth function, i.e. it should be differentiable for every value of x. However, the minimum

Algorithm 1 Minimize $F(x)$ using Gradient Descent

```
1: $x$ = initial solution
2: repeat
3:     Calculate derivatives $\Delta x = \partial F / \partial x$
4:     Choose a proper $\beta$ parameter
5:     Update $x$: $x = x - \beta \Delta x$
6: until convergence
```

cut formulation contains the absolute value function which is not differentiable at its root. Bhusnurmath and Taylor [8] solved this problem using the interior point method with logarithmic barrier potentials. At each iteration they solved a smoothed version of the function and the solution is given as the initial solution for the next iteration where the function is not smoothed as much as the previous iteration. With each successive step of this operation, the minimized function approaches to the original non-differentiable function while the solution of the smoothed function approaches to the desired solution. We employ a similar strategy. However, instead of smoothing the function $F(x)$ of Eq. 1 directly, we expand the Eq. 1 as

$$F(\mathbf{v}) = \sum_{i}^{n} \left(s_i |1 - v_i| + t_i |v_i| + \sum_{j \in N(i)} w_{ij} |v_i - v_j| \right) \tag{2}$$

and approximate the absolute value function by smoothing it with a small variable μ using

$$|v_i| = \sqrt{v_i^2 + \mu}\ . \tag{3}$$

Using this approximation of the absolute value function, Eq. 2 is now differentiable and solvable using the gradient descent method. The advantage of this approach is that expanded formulation requires less memory and provides more parallelization. The derivative calculation is local and highly parallel. The step factor β of Algorithm 1 is chosen at each iteration such that the update in the given direction is as much as possible while keeping the vector $\mathbf{v}$ feasible, i.e. $v_i \in [0, 1]$. Note that $\mathbf{v}$ does not have to be strictly in the interval $[0, 1]$ because it will converge eventually. However, we observed in our experiments that, this constraint dramatically speeds up the convergence.

The time complexity of our algorithm is the number of iterations times O(n) for a grid graph which is the time to compute the gradient direction and to update the variables. The memory complexity is linear in number of nodes for a grid graph.

As the gradient descent method employs local computation of derivatives and local update of variables, the propagation of information from one end of the graph to the other end might take longer for some applications. This results

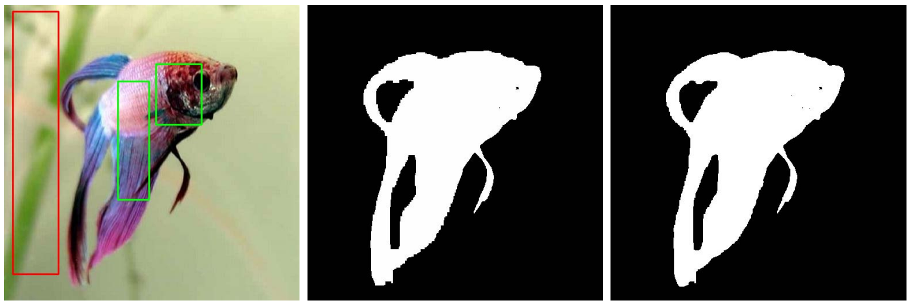

(a) Input image (b) Max-flow result (c) Our result

Fig. 2. Results for the fish image.

in higher number of iterations for some difficult examples. As the time required for a single iteration of the algorithm is less than the Newton's method, the total running time is expected not to be worse than the Newton's method of [8]. The advantage of gradient descent is the less memory requirement and the local management of variables. Another advantage is the possibility of applying the well-known popular parallel processing techniques such as the red-black ordering and the SOR. The red-black ordering method is the computation of independent variables in each iteration. Consider a 2D grid graph, which is very common in vision problems. If we think the graph as a checker board, the red and black squares are two sets of locally independent variables. Computing for the red variables and updating them will let the black variables use the most recent values of their neighbors and vice versa. This method dramatically decreases the number if iterations required for convergence. However the current Nvidia GPUs has a memory bottleneck unless the memory access pattern is well-organized. Although the number of iterations is decreased, the total load stays the same. We observed that red-black ordering is currently not improving the running times. On the other hand, applying SOR with ω around 1.5 speeds-up the convergence around 20-30%.

4 Experiments

We have tested our algorithm on the image segmentation task with various examples. The source and sink links are computed using mixtures of Gaussian built from the user selected sample regions. The neighboring link are computed using the images' spatial gradients. Figure 2 and 3 show a typical output of the segmentation task. To visually compare the results with the popular maximum flow algorithm of Boykov and Kolmogorov [1], we give outputs from both of the algorithms. As our algorithm smooths the energy function at each iteration with decreasing strengths, the borders of the segmentation seem to be smoother in our results which is desirable in some applications.

It can be seen in flower1 and puppy images that, our method produces some small erroneous regions which correspond to the graph regions that are difficult to solve. The flower2 image in Fig. 4 is especially a very difficult example and our method produces some erroneous regions. This is due to the approximation and smoothing of the energy. On the other hand, for very easy examples such as the flower3 image in Fig. 5, our method converges very fast (see Table 1) and gives very accurate results.

Table 1. Running times (in ms) comparison for max-flow and our algorithm with different images

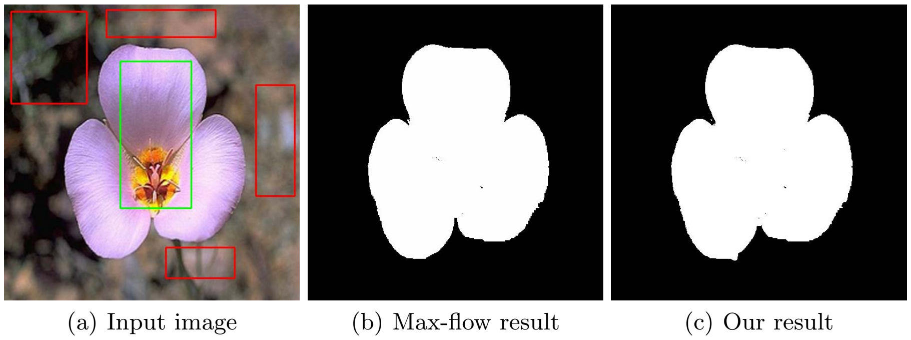
Image	Resolution	Max-flow [1]	Our method
flower1	512x512	111.14	112.5
flower2	512x512	128.64	194.63
flower3	512x400	72.92	8.04
puppy	512x372	82.44	94.39
fish	256x256	24.44	18.59
fish	512x512	128.56	117.6
fish	1024x1024	557.46	743.98

As noted before, our algorithm requires less memory than other graph cut algorithms. In Table 2, we give the memory usages of the max-flow algorithm of Boykov and Kolmogorov [1] and our gradient descent algorithm. Both algorithms' memory complexity seems linear, however, our method uses less memory compared to [1].

Table 2. Memory usage (in KBs) comparison for max-flow and our algorithm with different image sizes

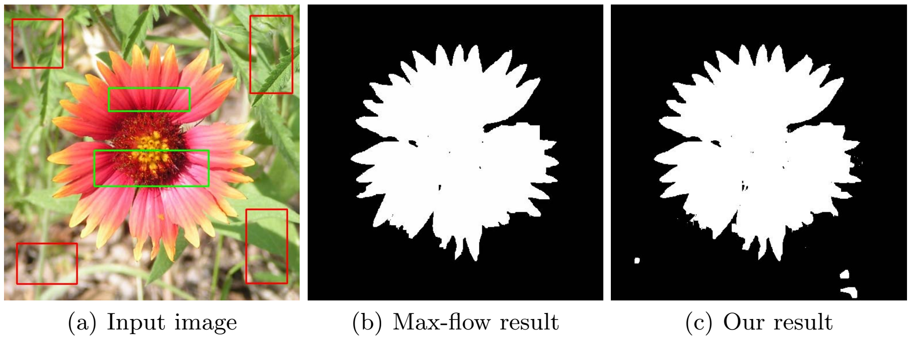
Algorithm	256x256	512x512	1024x1024
Max-flow [1]	6140	24672	98724
Our method	1536	6144	24576

Our gradient descent formulation can directly initiate from a given labeling without any processing in contrast to the maximum flow algorithms. For a video sequence, as the successive frames will have little difference in results, this property can be exploited very usefully. In Fig. 8 we give the comparison of running times of our algorithm with the max-flow algorithm [1]. The running times are produced from the 320x240 video sequence given in Fig. 7.



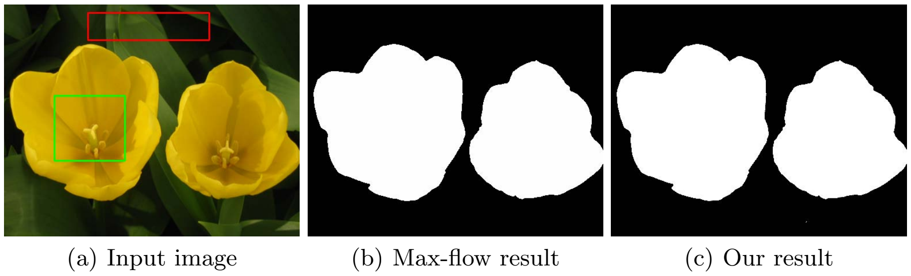
(a) Input image (b) Max-flow result (c) Our result

Fig. 3. Results for the puppy image.



(a) Input image (b) Max-flow result (c) Our result

Fig. 4. Results for the flower2 image.



(a) Input image (b) Max-flow result (c) Our result

Fig. 5. Results for the flower3 image.

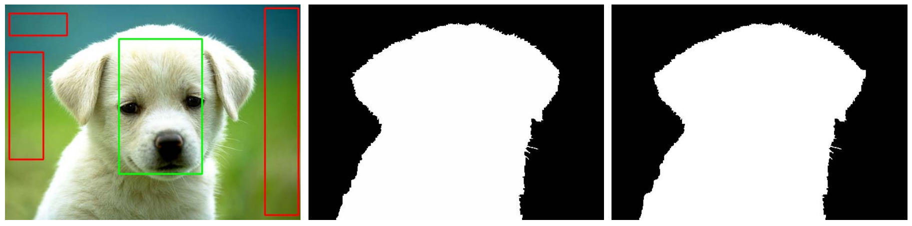

(a) Input image (b) Max-flow result (c) Our result

Fig. 6. Results for the puppy image.

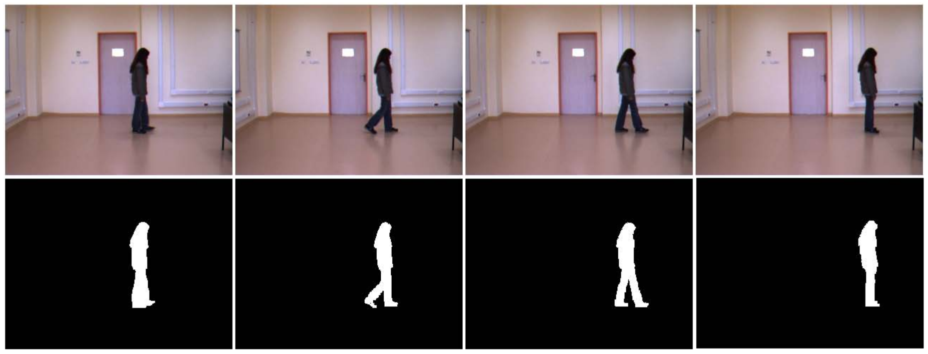


Fig. 7. Frames from a video sequence. The output of each frame is given as the initial segmentation for the next frame.

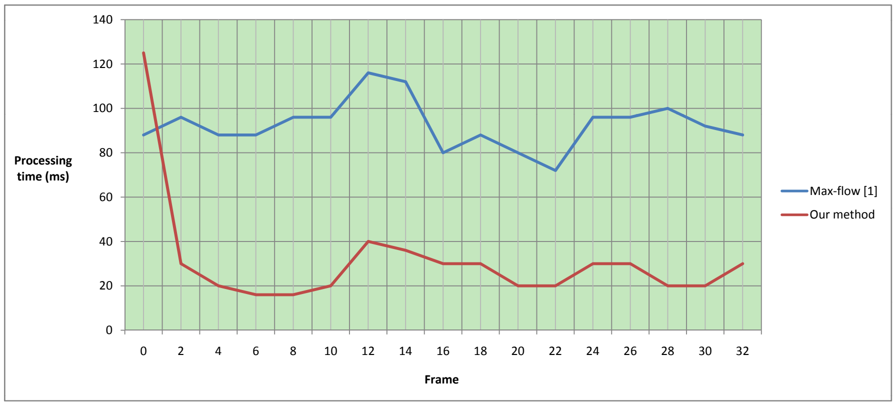


Fig. 8. Running times (in ms) for the video sequence given in Fig. 7.

5 Conclusions

We presented a gradient descent approximation of graph cuts, which is based on the unconstrained l_1-norm formulation of minimum cut. The advantages of our method are the local computation of derivatives and lower memory requirement. We also apply some well-known speed-up techniques such as the red-black ordering and SOR. We showed that our method can be easily initiated from a given approximation without any preprocessing on the graph, in contrast to dynamic maximum flow methods which initiate from a residual graph. The disadvantage of our method is the high number of iterations required due to the nature of the gradient descent algorithm. We address this issue using speed-up techniques and fully utilizing the parallel processors on the GPU. As a future work, we plan to investigate different strategies including the multi-scale scheme and heuristics to speed-up the method even further.

6 Acknowledgements

This work was conducted at the Computer Vision Laboratory at Gebze Institute of Technology. It was partially supported by TUBITAK Career Project 105E097.

References

1. Boykov Y. and Kolmogorov V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. PAMI 26(9) September 2004
2. GoldBerg A.V. and Tarjan R. E.: A new approach to the maximum flow problem. JACM 1988
3. Lombaert H., Sun Y., Grady L. and Xu C.: A multilevel banded graph cuts method for fast image segmentation. ICCV 2005
4. Juan O. and Boykov Y.: Active Graph Cuts. CVPR 2006 p.1023-1029
5. Kohli P. and Torr P. H. S.: Dynamic graph cuts for efficient inference in markov random fields. PAMI 2007
6. Dixit N., Keriven R. and Paragios N.: GPU-Cuts: Combinatorial optimisation, graphic processing units and adaptive object extraction. Research Report 05-07 March 2005 CERTIS
7. Vineet V. and Narayanan P. J.: CUDA Cuts: Fast graphs cuts on the gpu. Technical Report, International Institute of Information Technology, Hyderabd
8. Bhusnurmath A., Taylor C. J.: Solving the graph cut problem via l1-norm minimization. Technical Reports (CIS) 2007
9. Ford L. R. Jr. and Fulkerson D. R.: Maximal flow through a network. Canadian Journal of Mathematics, 1956
10. Nvidia Geforce Family GPUs, http://www.nvidia.com/object/geforce_family.html