Fast Graph Cuts using Shrink-Expand Reparameterization

Parikshit Sakurikar and P. J. Narayanan
Center for Visual Information Technology
International Institute of Information Technology, Hyderabad, India
{parikshit_s@students.,pjn@}iiit.ac.in

Abstract

Global optimization of MRF energy using graph cuts is widely used in computer vision. As the images are getting larger, faster graph cuts are needed without sacrificing optimality. Initializing or reparameterizing a graph using results of a similar one has provided efficiency in the past. In this paper, we present a method to speedup graph cuts using shrink-expand reparameterization. Our scheme merges the nodes of a given graph to shrink it. The resulting graph and its mincut are expanded and used to reparameterize the original graph for faster convergence. Graph shrinking can be done in different ways. We use a block-wise shrinking similar to multiresolution processing of images in our Multiresolution Cuts algorithm. We also develop a hybrid approach that can mix nodes from different levels without affecting optimality. Our algorithm is particularly suited for processing large images. The processing time on the full detail graph reduces nearly by a factor of 4. The overall application time including all book-keeping is faster by a factor of 2 on various types of images.

1. Introduction

Many labelling problems in computer vision, such as image segmentation, image denoising and stereo correspondence can be modelled as problems of energy minimization over a Markov Random Field [2, 12]. Though optimal label assignment is an NP hard problem, polynomial time algorithms are available for many classes of MRFs. The minimum-cut of a graph defined over the MRF gives the global minimum solution if the energy functions are submodular. Graph cuts has become important for many computer vision problems as a result. Efficient implementation of graph cuts is essential for practically useful applications. Efficiency is particularly important when processing multi-million pixel images that are common today.

Reparameterizing the graph to initialize it closer to the solution has been used to speedup graph cuts [7]. This requires a valid flow defined on a similar graph of identical topology, which can be subtracted from the given graph without affecting its mincut. Kohli and Torr used the initial and final graphs from an earlier frame to reparameterize later frames of a video [7]. Finding a graph with a similar mincut is the key to getting good performance using reparameterization.

In this paper, we present the *shrink-expand reparameterization* to find a graph with a similar cut from the given graph itself. The given graph is simplified using one or more shrink steps. Corresponding expansion steps restore the original graph topology. We show how a mincut of the simplified graph can be used to derive a valid flow for accurate reparameterization. Shrinking and expanding can be done in different ways on graphs. We present a spatial-proximity based shrink operation and the *multiresolution cuts* algorithm using it for image segmentation. Our approach achieves a 2× speedup over regular graph cuts on the CPU and the GPU, which is critical when processing large images. We also present a method that can provide a trade-off between accuracy and speed of segmentation in a user-controllable manner. Our implementation, built over public code available for the CPU and the GPU, is available for download.

Prior Work: In computer vision, the global minimum of an energy function of the form $E(f) = \sum D_i X_i + \sum V_{ij} X_i(X_j)$ is often sought, where the dataterm $D_i X_i$ measures the cost of assigning a label $l \in L$ to a pixel $p \in P$ and the smoothness term $V_{ij} X_i(X_j)$ measures the cost of assigning combinations of labels to neighboring pixels. A graph with a node for each pixel plus two designated terminal nodes and edge weights depending on the energy values can be constructed for bilevel problems. The mincut of this graph gives the global minimum of the submodular energy function and partitions the nodes into sets S and T corresponding to the terminal nodes s and t respectively [3]. The maxflow algorithms used to solve the mincut problem include the augmenting path method by Ford-Fulkerson [4] and the push-relabel method by Goldberg-Tarjan [5].

Careful and efficient implementations of the Ford-

Fulkerson algorithm have been used very widely in computer vision. Boykov and Kolmogorov [3] improved the augmenting path method by maintaining two search trees starting from both the source node and the sink node, and reusing them while paths were saturated one by one to find the mincut. Schmidt *et al*. [13] presented an $O(n \log n)$ method for planar graphs. The push relabel algorithm has also been implemented on the GPU for improvements in speed [15].

In dynamic graph cuts [7], Kohli and Torr achieved a speedup in graph cuts over a frame of a video by reparameterizing the graph using the initial and final graphs of a similar, previous frame. Alahari *et al*. [1] and Komodakis *et al*. [8] extended this idea to multi-labelling problems. The active graph cuts [6] method exploits approximate solutions obtained using different means including hierarchical processing for faster convergence. Lombaert *et al*. extended the traditional pyramid processing to achieve faster graph cuts at the expense of the global optimum [9]. Sinop and Grady extended this idea further and ensured global optimality by explicitly maintaining a separate record of changes [14].

Our method is a generalization of the reparameterization step of dynamic graph cuts, using the mincut on a simplified version of the original graph. It is also related to active graph cuts in reusing a similar cut, but follows the standard maxflow algorithm after reparameterizing the graph with a valid flow. The key idea is the construction of a valid flow using a number of graph shrinking and expansion operations.

2. Shrink-Expand Reparameterization

Shrink-Expand reparameterization to find the mincut of a given positive weighted, general graph $G = (V, E)$ involves creating a simplified graph G_1 from G using a single shrinking step. G_1 and its mincut can be used to find the mincut of G faster. We describe the generic *shrink* and *expand* steps essential to the reparameterization scheme.

Graph Shrinking: Consider a set A of k nodes of G as shown in Figure 1. Set A doesn't include nodes s or t. A graph $G_1 = (V_1, E_1)$ is formed by merging nodes of A into a single node v_A. Nodes of A are replaced by v_A in every edge. Self-edges that may result from this are omitted. Weights of resulting mulitedges are replaced by their average value. This step resembles the supervertex formation used by minimum spanning tree and connected component algorithms [11]. The shrinking process is defined formally below (Figure 1).

- V_1 has all nodes of $(V - A)$ and a single node v_A for all nodes in A.
- Edges $(u, v) \in E$ where $u, v \notin A$ are copied to E_1.
- Each edge $(u, v) \in E$ where $u \in A$ and $v \notin A$ is replaced in E_1 by a single edge (v_A, v) with the same weight. If several $u \in A$ are connected to node v, the average of all such edges is assigned to (v_A, v).
- Edges between nodes of A in G do not appear in G_1.

Graph Expansion: A graph $G_1 = (V_1, E_1)$ can be expanded to yield a graph $\hat{G}_1 = (\hat{V}_1, \hat{E}_1)$. Expansion is defined only with respect to a shrinking step using a set A. It restores all nodes and edges of the graph G that yielded G_1. Edges that were averaged during shrinking separate and get identical weights equal to the corresponding edge in E_1. The omitted edges between nodes of A all get zero (or ϵ) weights. Thus, graphs G and the $\hat{G}$ have identical topology, but may have different edge weights. The expansion process is defined formally below (Figure 1).

- $\hat{G}_1$ has all nodes of G_1, with node v_A being replaced by the set of nodes in A. It has all edges of G, including the edges between nodes of A and the rest of the graph saved during the shrinking step.
- Weights of edges $(u, v) \in E_1$ where $u, v \neq v_A$ are copied to corresponding edges of $\hat{E}_1$.
- Weights of edges $(u, v_A) \in E_1$ are copied to every edge in $\hat{E}_1$ between u and nodes of A.
- Edges within nodes of A have an infinitesimal weight in $\hat{E}_1$.

Augmented Expansion: Let C be the mincut of G_1, with the corresponding residual graph G_1^r. The edges of G_1^r have reduced weights. The cut C partitions the nodes of G_1 into sets S and T which are reachable from nodes s and t respectively in G_1^r. These correspond to sets $\hat{S}$ and $\hat{T}$ in $\hat{G}_1$. We create an *augmented graph expansion* ${}^a\hat{G}_1$ with respect to a cut C by adding an edge of arbitrarily high weight between each node and a terminal node. An edge from s to a node $\hat{u}$ is added in $\hat{E}_1$ if $\hat{u} \in \hat{S}$ based on the cut C. An edge from $\hat{u}$ to t is added if $\hat{u} \in \hat{T}$. All nodes of A are connected to s if $v_A \in S$ and to t otherwise.

Claim 1: The graphs ${}^a\hat{G}_1$ and ${}^a\hat{G}_1^r$ have the same mincut.

Proof: We show that ${}^a\hat{G}_1$ can be converted to ${}^a\hat{G}_1^r$ by pushing a few valid flows. Let $\hat{S}$ and $\hat{T}$ be the set of nodes in ${}^a\hat{G}_1$ that are expanded from nodes of the source and target sets S_1 and T_1 respectively in G_1. There are no edges in ${}^a\hat{G}_1^r$ between nodes of $\hat{S}$ and $\hat{T}$ since there are none in G_1^r. ${}^a\hat{G}_1^r$ is hence a residual graph. Also, the augmented graphs have large weighted edges from s to the nodes in $\hat{S}$ and from the nodes in $\hat{T}$ to t by construction. Consider an edge $(\hat{u}, \hat{v}) \in {}^a\hat{G}_1$ with $\hat{u} \in \hat{S}$ and $\hat{v} \in \hat{T}$. This edge can be saturated in one step by pushing flow

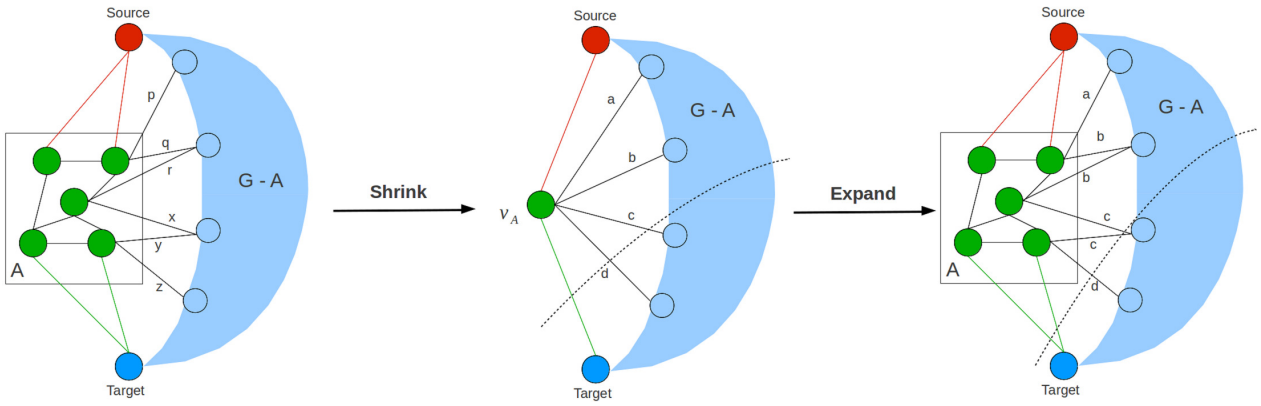

Figure 1: Shrink-Expand Reparameterization: Graph G (left), with a set of nodes A, gives graph G_1 (middle) where the set A is merged to form the node v_A. The edge a has the same weight as the edge p. The edge b is the average of edges q and r, and so on. The expansion of G_1 gives $\hat{G}_1$ (right), which has the same topology as that of G. The dotted line indicates the cut of the graph G_1 which is subsequently reflected in the expanded graph $\hat{G}_1$. Augmented Expansion adds high weight edges in $\hat{G}_1$ from each node to s or t.

equal to its capacity from s to $\hat{u}$ to $\hat{v}$ and then to t. The graph ${}^a\hat{G}_1$ transforms to ${}^a\hat{G}_1^r$ if all such edges are saturated.

Claim 2: The reparameterized graph $\tilde{G} = G - (\hat{G}_1 - \hat{G}_1^r)$ has the same mincut as G, where $A - B$ between graphs of same topology refers to the subtraction of edge weights of B from corresponding edge weights of A.

Proof: Since ${}^a\hat{G}_1$ and ${}^a\hat{G}_1^r$ have the same mincut, the graph $\tilde{G} = G - ({}^a\hat{G}_1 - {}^a\hat{G}_1^r)$ will be a reparameterization of the graph G in which the mincut is preserved as shown by Kohli and Torr [7]. Also, $({}^a\hat{G}_1 - {}^a\hat{G}_1^r) = (\hat{G}_1 - \hat{G}_1^r)$, as the augmented edges cancel out in the subtraction due to their very high weights. Thus, $\tilde{G}$ has the same mincut as G.

Shrinking Steps for Images: Shrink-Expand reparameterization is a framework to generate graphs with the same mincut as that of a given graph. The mincut is preserved when a number of shrink steps are performed together or sequentially. The cut on a reparameterized graph is likely to be faster as the graph $\tilde{G}$ is closer to the final residual graph. Different ways of grouping nodes in the shrink step can be used. In image processing, node grouping can be based on pixel similarity as in superpixels or pixel proximity. On general graphs, nodes could be clustered based on other neighbourhood properties. We now present *multiresolution reparameterization*, inspired by pyramid image processing, that uses pixel proximity for grouping.

3. Multiresolution Reparameterization

For multiresolution shrink and expand operations, the MRF graph over the full resolution image is formed first. Every fixed, non-overlapping $m \times m$ block of graph nodes is combined into a single node in a shrink step to yield the graph G_1 with far fewer nodes. The mincut of G_1 is computed before applying expansion to yield $\hat{G}_1$, $\hat{G}_1^r$, and $\tilde{G}$ which yields the final mincut of G. Similar reparameterization can be applied to compute the mincut on G_1 by forming a graph G_2, G_2^r, and $\tilde{G}_1$ at lower levels. This process can be repeated for several levels for large images, if it is profitable to do so. This *multiresolution cuts* algorithm uses this reparameterization to compute the global minimum of the original graph G. It differs from banded graph cuts [9, 14] and the hierarchical segmentation based active graph cuts [6] as our apparent resolution change is in the graph space only. We do not build image pyramids or graphs at each pyramid. The nodes of the graph built from the original image are combined to form graphs of lower sizes. This gives us the flexibility to combine multiple levels as discussed later.

3.1. Multiresolution Cuts

Our method consists of shrinking the graph by grouping $m \times m$ blocks of nodes together and creating edges with averaged weights (Figure 2). Expansion reverses this process and creates a graph with zero weight edges within the

Algorithm 1: Multiresolution Cuts

1: Build full resolution graph G_0 on the input image
2: Generate graphs $G_1, G_2, \cdots G_L$ by shrinking $m \times m$ blocks of nodes into a single node in each step.
3: Perform graph cuts on G_L to get its mincut and G_L^r
4: **for** $i = (L-1) \rightarrow 0$ **do**
5: Construct $\hat{G}_{i+1}, \hat{G}_{i+1}^r$ and $\tilde{G}_i$
6: Perform graph cuts on $\tilde{G}_i$ to get its mincut and G_i^r
7: **end for**

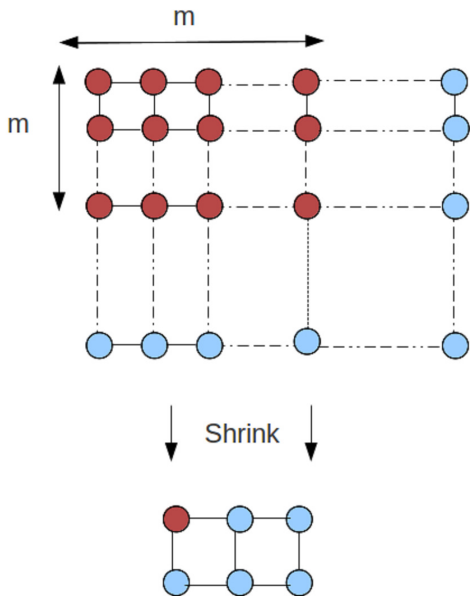

Figure 2: Multiresolution Cuts: Each non-overlapping block of $m \times m$ nodes is merged to form one node in the lower resolution graph. Expansion reverses this process and gives equal weights to edges between different blocks.

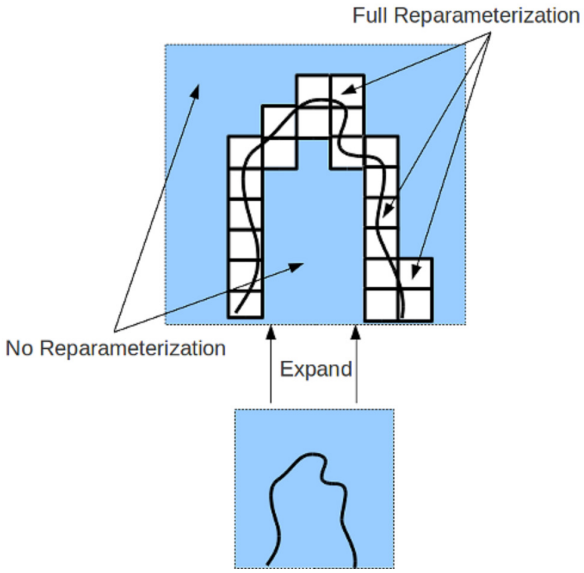

Figure 3: Hybrid Reparameterization: The large, smooth regions of the image in the background and the foreground are not reparameterized on expansion, thus reducing the time that the algorithm spends in processing these regions.

blocks and equal weight edges between nodes of adjacent blocks. The performance of this approach depends on the block size m and number of levels used. For large values of m, the graph shrinks fast with quick cuts at the shrunk levels. However, the reparameterized graph is farther from the original graph at upper levels which increases the processing time. Experimentally, we found shrinking 4×4 blocks of nodes to work the best for large graphs. Algorithm 1 decribes the *multiresolution cuts* process in detail.

The number of levels to be used depends on the benefits of shrinking further compared to performing the cut at the same level. We found that performing the cut directly is faster when the graph has approximately 4K-8K nodes when using Boykov's graph cuts code on the CPU. This translates to 4 levels of 4×4 reduction for a 16MP image. The corresponding number when using the NPP code on the GPU [10] is about 10K-16K nodes, below which the single level cut is cheaper than using reparameterization.

3.2. Hybrid Reparameterization

Global optimality is guaranteed if the graph G is reparameterized by a valid flow to give $\tilde{G}$. Shrink and expand steps can proceed at their own pace as long as this is satisfied. This can be exploited to create graphs with mixed resolution levels, with larger regions combined into single nodes in smooth regions. We do this based on the mincut at the highest shrink (or lowest resolution) level. Regions near the cut edges are expanded to the next level while regions far from it skip some of the intermediate levels (Figure 3). We skip a level i of expansion for a region by using weights from the residual graph $\hat{G}^r_{i+1}$ for the subgraph resulting from that region. Regions that are not skipped are reparameterized normally in $\tilde{G}_i$ by subtracting valid flows from level $(i+1)$. This results in the mincut algorithm doing practically no work in the skipped region but provides a pair of $\tilde{G}_i$ and G^r_i pairs that can be used to reparameterize the next level $(i-1)$. It should be noted that the cut of the hybrid $\tilde{G}_i$ is not equal to the cut of graph G_i. Complete reparameterization at the most detailed level guarantees global optimum.

3.3. Minimally Approximate Cut

The final cut of $\tilde{G}$ at the highest level takes the most time in all examples. We can trade computation time for some approximation of the results by modifying this step slightly. We follow normal procedure for levels 1 onwards. In the final step, $\hat{G}_1$ and $\hat{G}^r_1$ are restricted to just around the bounding box of the foreground object in G_1, skipping regions outside of it. This limits the nodes that take part in the final graph cuts, similar to the banded graph cuts approach. This method provides further speedup when foreground objects are relatively compact. The exact boundary may not be obtained. The discrepancy in segmentation affects fewer than 0.02% of pixels in our experience, for a further saving in time of 20-25%.

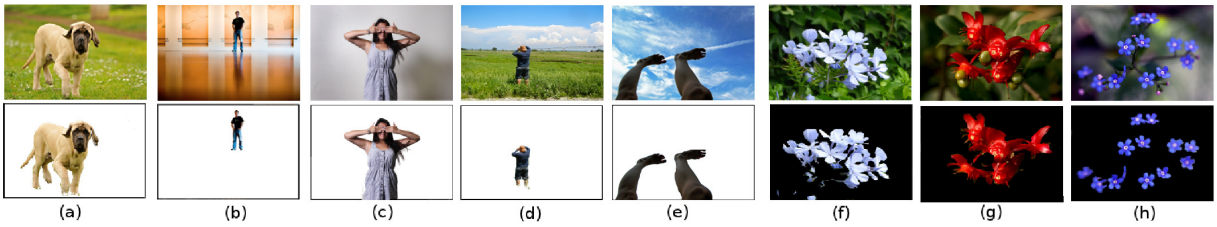

Figure 4: Illustration of Graph Cut on a few images of varying sizes. Runtimes given in Tables 1 and 3. (a) 2 Mpixel image (b) 4 Mpixel image (c) 8 Mpixel image (d) 12 Mpixel image (e) 16 Mpixel image (f) 8 Mpixel image with distributed foreground segments. (g) 12 Mpixel image, distributed foreground. (h) 16 Mpixel image, distributed foreground. All images obtained from the Creative Commons section of Flickr.

4. Experimental Results

We present the results of using graph cuts to segment several large images on the CPU and on a GPU. The experiments are performed on an Intel Core i7 CPU. The GPU experiments are performed on an Nvidia GTX-580 with 512 cores using CUDA4.0 library. The MRF graph weights are assigned according to the GrabCut method [12]. The time to learn the GMMs is not included in the timings. We present the total application time unless mentioned otherwise. The total application time includes the time taken for graph construction, shrink-expand manipulation, reparameterization and the graph cuts optimization steps. We concentrate on processing large images for which computational efficiency is more important. Thus, we show results on images with 2 or more millions of pixels (MP). We also distinguish between images with relatively compact foreground and those with distributed foreground. We extend the code by Boykov and Kolmogorov (BK) for the CPU implementation [3]. On the GPU, we extend the code from the NPP library of CUDA 4.0 provided by Nvidia [10].

Table 1 gives the times for different stages of our algorithm for two typical 16MP images using four levels of 4×4 shrinking. The graph cut on the final reparameterized graph $\tilde{G}$ on the CPU, is about 4 times faster than the graph cut on the original graph. The time to compute the cuts at all the other levels is significantly smaller than this. The application time though is only 2 times faster than the standard BK application time due to the overhead of the shrink, expand, and reparameterization steps. Our speedup at different levels exceeds the speedup reported by the hierarchical segmentation based active graph cuts [6] method.

Figures 5a & 5b show the variation of the application time with number of levels for images of several sizes. The time shown for 0-level corresponds to graph cuts without our approach. Each level represents a shrinking of the image using 2×2 blocks. The total time reduces with the number of levels until a minimum value and increases thereafter. The experiments show that the best performance is obtained when the coarsest graph has 4K to 8K nodes, as direct cut is faster at that size or lower. The minimum point is 10K to 16K nodes when using the GPU as it can process larger images faster.

Figures 5c & 5d show the variation in application time of computing graph cuts using 2×2 and 4×4 shrink blocks for a 12 MP image. The 4×4 scheme of downsampling works marginally better among the different schemes we tried, being a balance between the number of levels and the quality of reparameterization. We use this scheme for our experiments.

We experimented with hybrid reparameterization on both the CPU and the GPU platforms. During the expand

Graph Resolution	Fig 4e		Fig 4h	
	CPU(s)	GPU(ms)	CPU(s)	GPU(ms)
Direct Cut (16M)	7.91	761	10.27	959
Multiresolution Cuts:				
Level 0 (Full = 16M)	2.01	219	2.83	257
Level 1 (1M)	0.30	118	0.51	113
Level 2 (64K)	0.06	54	0.17	79
Level 3 (4K)	0.01	–	0.04	–

Table 1: Time taken for only the graph cuts step at different stages for two 16 MP images.

	Graph Cuts	Our Method		MAC	
		2×2	4×4	2×2	4×4
	Compact foreground (Fig 4e)				
CPU (sec)	9.9	5.3	4.9	3.6	3.2
GPU (ms)	985	508	461	323	269
% Err pixels		0	0	0.005	0.012
	Distributed foreground (Fig 4h)				
CPU (sec)	12.2	6.9	6.1	6.0	5.9
GPU (ms)	1049	578	531	518	471
% Err pixels		0	0	0.002	0.006

Table 2: Application time results of minimally approximate cuts for 16MP images.

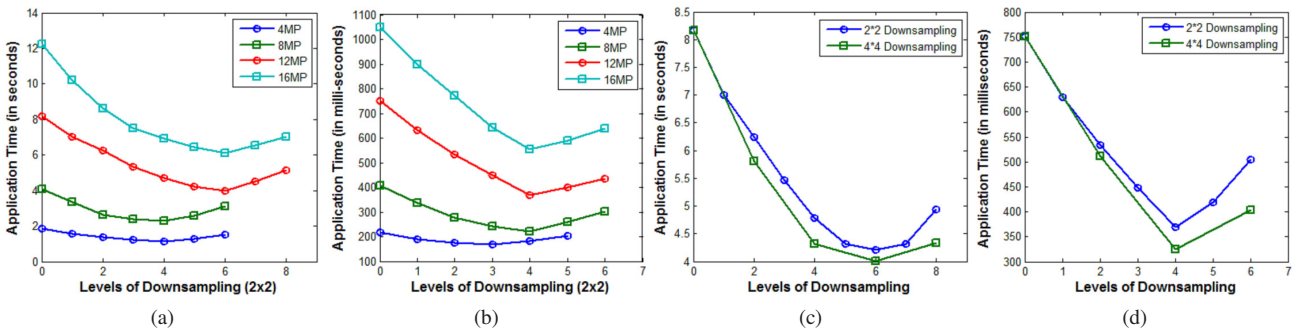

Figure 5: (a) Shows variation of application time with number of levels of downsampling(shrink) on the CPU. (b) shows the corresponding results on the GPU. (c) & (d) show the variation of application time with shrink resolution on the CPU and GPU respectively for a 12MP image (4g).

operations of this scheme, only those blocks of $m \times m$ nodes which include a boundary pixel from the cut are reparameterized normally. All other blocks are are skipped as explained in Section 3.2. Thus at intermediate levels, the algorithm spends no time in processing large smooth regions of the background and the foreground. The gains due to the hybrid approach are significant at lower levels on the CPU, but the final cut at the full resolution level takes more time than the multiresolution method, leading to no overall improvement in application time. On the GPU, the hybrid scheme reduces the application time by about 10-15%. This difference is perhaps due to their implementations[1]. We thus use Multiresolution Cuts on the CPU and Hybrid Multiresolution Cuts on the GPU. For a 16 MP image (Figure 4e), the application time for Hybrid Multiresolution Cuts is 5.73 seconds and 461 ms respectively on the CPU and GPU as compared to 4.99 seconds and 509 ms for Multiresolution Cuts.

Table 3 summarizes the application times on the CPU and the GPU for many images including all from Figure 4. In general, the running time increases with the number of pixels in the cut perimeter. The average speedup of our approach is 2 on the CPU and the GPU. Our approach consistently produces this speedup on compact and distributed foreground images. Active graph cuts using hierarchical segmentation obtains no speedup when the foreground segment is complicated as reported in [6]. The GPU performs 10 times faster than the CPU on the whole in both approaches.

Table 2 gives the results of the minimally approximate cuts approach. We restrict the last level cut to the bounding box of the foreground of G_1. This improves the speedup significantly for compact foregrounds with fewer than 0.02% misclassified pixels. The saving is not much for distributed foregrounds as only a few pixels are eliminated at the last level. This method can result in faster graph cuts if errors can be tolerated to a certain extent.

[1]The BK code performs significant preprocessing over the built graph for performance, adding a large fixed time to all graph cut computations. For instance, if one feeds the final residual graph of a previous graph cut operation as the input graph, the BK code takes about 33% of the time to converge as it did for the original graph though no flow occurs. The residual graph converges in negligible time – well under 2% of total time – using the GPU code.

5. Conclusions and Discussion

We presented a shrink-expand reparameterization scheme to speedup graph cuts in this paper. We demonstrated its effectiveness using multiresolution shrinking for grid graphs used in computer vision. Our approach preserves the global minimum. Our implementation is built over standard code of graph cuts on the CPU and the GPU. It can hence benefit from improvements in each. We achieve a speedup of about 4 for the final graphs and about 2 for the overall graph cut application. We also presented hybrid reparameterization of the graphs based on an adaptive method of construction of the mixed-level graphs which can improve efficiency. The approximate version we presented can achieve further speedup using an adaptive scheme at the highest level.

Other ways to shrink the graph may be useful in other situations. For example, shrinking nodes based on local similarity of corresponding image pixels creates a multilevel, superpixel-based segmentation scheme. The boundaries within the superpixels will be resolved at later levels of the algorithm. Stopping at a less detailed level will yield coarse segmentations that are not blocky, unlike the multiresolution cuts scheme. The graphs at different shrink levels may not have a simple mesh structure. This can result in lack of efficiency using standard implementations like BK and NPP. The basic method works even if distant nodes are combined in the shrink step. However, the overall efficiency will depend on how close the cut at the simplified level is to the expanded level.

Image Size (MPixels)	Foreground (% pixels)	Cut Perimeter (% pixels)	Time on CPU (sec)			Time on GPU (millisec)		
			BK	MC	Speedup	NPP	HMC	Speedup
Compact Foreground								
2 (4a)	37	0.34	1.60	0.92	1.74	126	88	1.43
2	26	0.22	0.93	0.55	1.70	83	52	1.59
4 (4b)	2.5	0.05	1.86	1.11	1.67	216	151	1.43
4	11	0.10	2.39	1.35	1.76	226	153	1.47
8 (4c)	28	0.18	4.10	2.29	1.79	407	211	1.92
8	19	0.11	3.93	2.00	1.95	412	223	1.84
8	7	0.06	2.61	1.47	1.78	313	155	2.01
12 (4d)	3.6	0.08	7.34	3.89	1.88	678	337	2.01
12	9	0.17	7.49	3.93	1.90	686	342	2.00
12	16	0.13	8.20	4.15	1.97	713	351	2.03
16	5.1	0.09	7.48	3.38	2.20	752	357	2.10
16	12	0.11	8.35	3.98	2.09	882	459	1.92
16 (4e)	21	0.23	9.94	4.99	1.98	985	461	2.13
Distributed Foreground								
8 (4f)	38	0.71	5.87	3.10	1.89	453	210	2.15
12 (4g)	35	0.74	8.17	4.00	2.04	751	325	2.31
16 (4h)	24	0.58	12.24	6.10	2.00	1049	531	1.97

Table 3: Comparison of BK with Multiresolution Cuts (MC) on the CPU and of NPP with Hybrid Multiresolution Cuts (HMC) on the GPU.

Shrink-expand reparameterization works on graphs of arbitrary structure also. The shrinking algorithm should depend on the properties of the nodes and edges of the graph. A shrink step similar to the supervertex formation of connected component algorithms or spanning tree algorithms can help find layered cuts in the original graph [11]. We expect to explore several combinations of the shrink step for efficient computation of mincuts of general graphs in the future.

References

[1] K. Alahari, P. Kohli, and P. H. S. Torr. Reduce, reuse & recycle: Efficiently solving multi-label MRFs. In *CVPR*, 2008. 66

[2] Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *ICCV*, pages 105 –112, 2001. 65

[3] Y. Boykov and V. Kolmogorov. An experimental comparison for min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on PAMI*, 26(9):1124–1137, 2004. 65, 66, 69

[4] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, Princeton, N.J.,, 1962. 65

[5] A. V. Goldberg and R. E. Tarjan. An new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. 65

[6] O. Juan and Y. Boykov. Active graph cuts. In *CVPR (1)*, pages 1023–1029, 2006. 66, 67, 69, 70

[7] P. Kohli and P. H. S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *IEEE Transactions on PAMI*, 29:2079–2088, 2007. 65, 66, 67

[8] N. Komodakis and G. Tziritas. Fast approximately optimal solutions for single and dynamic MRFs. In *CVPR*, 2007. 66

[9] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *ICCV*, pages 259–265, 2005. 66, 67

[10] P. J. Narayanan, V. Vineet, and T. Stitch. Fast graph cuts on the GPU. In *GPU Computing Gems*, volume 1, chapter 29, pages 439–450. Morgan Kaufmann, Dec. 2010. 68, 69

[11] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, pages 1389–1401, Nov. 1957. 66, 71

[12] C. Rother, V. Kolmogorov, and A. Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, 23:309–314, 2004. 65, 69

[13] F. R. Schmidt, E. Töppe, and D. Cremers. Efficient planar graph cuts with applications in computer vision. In *CVPR*, pages 351–356, 2009. 66

[14] A. K. Sinop and L. Grady. Accurate banded graph cut segmentation of thin structures using laplacian pyramids. In *MICCAI 2006*, volume II of *LNCS 4191*, pages 896–903, Oct. 2006. 66, 67

[15] V. Vineet and P. J. Narayanan. Cuda cuts: Fast graph cuts on the GPU. *CVPR Workshop on CVGPU*, pages 1–8, 2008. 66