1-1-2007

Solving the Graph Cut Problem via /1 Norm Minimization

Arvind Bhusnurmath
University of Pennsylvania, bhusnur4@seas.upenn.edu

Camillo J. Taylor
University of Pennsylvania, cjtaylor@cis.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-10.

Solving the Graph Cut Problem via /1 Norm Minimization

Abstract

Graph cuts have become an increasingly important tool for solving a number of energy minimization problems in computer vision and other fields. In this paper, the graph cut problem is reformulated as an unconstrained /1 norm minimization. This /1 norm minimization can then be tackled by solving a sequence of sparse linear systems involving the Laplacian of the underlying graph. The proposed procedure exploits the structure of these linear systems and can be implemented effectively on modern parallel architectures. The paper describes an implementation of the algorithm on a GPU and discusses experimental results obtained by applying the procedure to graphs derived from image processing problems.

Disciplines

Theory and Algorithms

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-07-10.

Solving the Graph Cut Problem via l_1 Norm Minimization

Arvind Bhusnurmath
GRASP Lab
CIS, University of Pennsylvania
bhusnur4@seas.upenn.edu

Camillo J Taylor
GRASP Lab
CIS,University of Pennsylvania
cjtaylor@cis.upenn.edu

Abstract

Graph cuts have become an increasingly important tool for solving a number of energy minimization problems in computer vision and other fields. In this paper, the graph cut problem is reformulated as an unconstrained l_1 norm minimization. This l_1 norm minimization can then be tackled by solving a sequence of sparse linear systems involving the Laplacian of the underlying graph. The proposed procedure exploits the structure of these linear systems and can be implemented effectively on modern parallel architectures. The paper describes an implementation of the algorithm on a GPU and discusses experimental results obtained by applying the procedure to graphs derived from image processing problems.

1. Introduction

Over the past decade graph cuts have emerged as an important tool for solving a number of energy minimization problems encountered in computer vision and machine learning. In their seminal paper, Kolmogorov and Zabih [15] show that any energy function that satisfies a property called regularity can be minimized by finding the minimum cut of a graph whose edge weights are related to the energy function. The energy functions that are encountered in many computer vision problems satisfy this condition which helps to explain the popularity of the approach.

Problems like image restoration [5], segmentation [3, 20] and stereo [10, 14, 21] have all been reduced to graph cut problems. Figure 1 shows the typical structure of the resulting graphs. Here the nodes s and t correspond to class labels while the interior nodes correspond to the pixels in the image. The graph cut methodology can also be applied to problems on 3D grids such as surface reconstruction [19, 22, 23].

Graph cut problems are usually solved using the equivalent maxflow formulation with Ford-Fulkerson or Push-relabel methods which can be found in standard algorithms textbooks such as Cormen *et al*. [7]. However, most of the graphs that are encountered in vision problems tend to have an extremely structured form - that of a grid. Boykov and Kolmogorov [4], exploit this fact and tune the Ford-Fulkerson algorithm to obtain better performance. The basic idea is to employ two search trees, one emanating from the source and one from the sink, which are updated over the course of the algorithm. Parallel implementations of the push relabel approach on a GPU have also been described by Dixit, Keriven and Paragios [8].

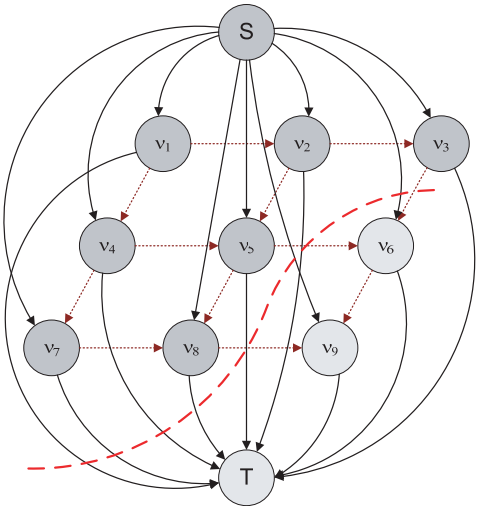

Figure 1. The figure shows the typical grid-like graph found in vision problems. The dotted curve indicates the min cut.

In many cases it is possible to use the solution obtained from some prior graph to efficiently compute the mincut of a similar graph. Kohli and Torr [13] use this idea while solving the object-background segmentation problem on video sequences. Instead of creating a graph and computing the mincut for each frame, the residual graph from the previous frame is updated with the few edge weights that change and a dynamic maxflow algorithm is used to compute the new maximum flow. Juan and Boykov [11] use an initial partition of the image pixels as a starting cut and then update this solution to obtain the final optimal cut. An initial non-

feasible flow is created from the initial cut and this solution is then driven towards feasibility. This method, known as Active Cuts can be employed in a coarse to fine scheme where the cut on a coarse level is used as an initial solution for a finer level. The scheme has also been applied to video segmentation.

Lombaert *et al*. [17] also solve image segmentation in a hierarchical manner by first finding the minimum cut on a coarser graph and then using this cut to build narrow banded graphs at higher resolution that consist of nodes around the boundary of the currently segmented objects. In other words, the coarser level provides an estimate of the segmentation and later stages are used to refine the location of the boundary.

This paper describes an alternative approach to solving the graph cut problem. Here the graph cut problem is reformulated as an unconstrained l_1 norm minimization problem. This convex optimization problem can be solved using the barrier method which effectively reduces the orginal optimization problem to the problem of solving a sequence of sparse linear systems involving the graph Laplacian. The resulting procedure can be implemented using vector operations and is well suited for implementation on parallel architectures.

The remainder of the paper is organized as follows: Sections 2 and 3 describe the underlying theory and the proposed implementation. Section 4 presents experimental results obtained with the procedure and Sections 5 and 6 discuss conclusions drawn and future work.

2. Theory

The goal of the min-cut problem is to divide the nodes in the graph shown in Figure 1 into two disjoint sets, one containing s and the other containing t, such that the sum of the weights of the edges connecting these two sets is minimized. In the sequel n will denote the number of interior nodes in the graph while m will represent the total number of edges. This min-cut problem is typically solved by considering the associated max-flow problem. That is, if we view the edges as pipes and the associated edge weights as capacities, we can consider the problem of maximizing the total flow between the source node, s, and the sink node, t, subject to the constraint that each of the interior nodes is neither a sink nor a source of flow. Like many combinatorial optimization problems [18], the max-flow problem can be expressed as a linear program as shown in Equation 1.

Let $\mathbf{x} \in \mathbb{R}^m$ denote a vector indicating the flow in each of the edges of the graph. A positive entry in this flow vector corresponds to a flow along the direction of the arrow associated with that edge, while a negative value corresponds to a flow in the opposite direction. In other words the edges in our graph are undirected and the associated arrows merely represent the convention used to interpret the flow values.

The goal of the optimization problem is to maximize the inner product $\mathbf{c^T x}$ where $\mathbf{c} \in \mathbb{R}^{\mathbf{m}}$ is a binary vector with +1 entries for all of the edges emanating from s and 0 entries elsewhere; this inner product effectively computes the net flow out of node s.

In order to express the constraint that the net flow associated with each of the interior nodes in the graph should be zero we introduce the node edge incidence matrix $A \in \mathbb{R}^{n \times m}$ whose rows and columns correspond to interior nodes and graph edges respectively. Each column of this matrix corresponds to an edge in the graph and will contain at most two non-zero entries, a +1 entry in the row corresponding to the node at the head of the arrow associated with that edge and a -1 for the node at the other end of the edge. Note that those columns corresponding to edges starting at s or terminating at t will only contain a single non-zero entry since the A matrix does not contain rows corresponding to the s and t nodes.

The product $A\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ denotes the sum of the flows impinging upon each of the interior nodes due to the flow assignment $\mathbf{x}$. The constraint $A\mathbf{x} = \mathbf{0}$ reflects the fact that the net flow at each of the interior nodes should be zero. The vector $\mathbf{w} \in \mathbb{R}^{\mathbf{m}}$ represents the non-negative weights associated with each of the edges in the graph. The inequalities $-\mathbf{w} \leq \mathbf{x}$ and $\mathbf{x} \leq \mathbf{w}$ reflect the capacity constraints associated with each of the edges.

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c^T x} \\ \text{st} \quad & A\mathbf{x} = \mathbf{0} \\ & -\mathbf{w} \leq \mathbf{x} \leq \mathbf{w}. \end{aligned} \tag{1}$$

A careful reader will note that this formulation differs slightly from the one presented by Zabih and Kolmogorov [15] which makes use of a directed graph. However, it can be shown that this formulation allows us to represent precisely the same set of objective functions as the ones described in that work.

Instead of tackling the linear program described in Equation 1 directly, we proceed by formulating the associated dual problem. More specifically, the Lagrangian of the linear program described in Equation 1 has the following form

$$\begin{aligned} L(\mathbf{x}, \lambda, \nu) &= -\mathbf{c}^T\mathbf{x} + \nu^{\mathbf{T}}\mathbf{A}\mathbf{x} + \lambda_+^{\mathbf{T}}(\mathbf{x} - \mathbf{w}) \\ &\quad + \lambda_-^T(-\mathbf{x} - \mathbf{w}) \\ &= -\mathbf{w}^T(\lambda_+ + \lambda_-) \\ &\quad + \mathbf{x}^T(A^T\nu - \mathbf{c} + \lambda_+ - \lambda_-) \end{aligned} \tag{2}$$

In this expression the original primal objective function is augmented with multiples of the constraint functions **(x - w)** ≤ 0, **(-x - w)** ≤ 0 and $A\mathbf{x} = 0$. There are three sets

of Lagrange multipliers $\lambda_+ \in \mathbb{R}^m$, $\lambda_- \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^n$ corresponding respectively to these three constraints.

The Lagrangian dual function $g(\lambda, \nu)$ is obtained by minimizing the Lagrangian with respect to $\mathbf{x}$ as shown below:

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \tag{3}$$

In this case the dual function has the following form.

$$g(\lambda, \nu) = \begin{cases} -\mathbf{w}^T(\lambda_+ + \lambda_-) & \text{iff } A^T\nu - \mathbf{c} = (\lambda_- - \lambda_+) \\ -\infty & \text{otherwise.} \end{cases} \tag{4}$$

We can compute the optimal value of our original primal problem by maximizing the associated Lagrangian dual function, $g(\lambda, \nu)$, which gives rise to the following dual problem.

$$\begin{aligned} \min_{\lambda,\nu} \quad & \mathbf{w}^T(\lambda_+ + \lambda_-) \\ \text{st} \quad & A^T\nu - \mathbf{c} = (\lambda_- - \lambda_+) \\ & \lambda_+ \geq 0,\ \lambda_- \geq 0, \end{aligned} \tag{5}$$

Here we note that the structure of the problem implies that the value of the inner product $\mathbf{w}^T(\lambda_+ + \lambda_-) = \sum_{i=1}^{m} \mathbf{w}_i(\lambda_+ + \lambda_-)_i$ will be minimized when $(\lambda_+ + \lambda_-)_i = |(A^T\nu - \mathbf{c})_i|$. To see this, consider that each term in this inner product corresponds to the weighted sum of two *non-negative* variables, λ_{-_i} and λ_{+_i} where their difference is constrained by $(\lambda_- - \lambda_+)_i = (A^T\nu - \mathbf{c})_i$. [1] In this case the minimum value that $(\lambda_{-_i} + \lambda_{+_i})$ can attain is $|(A^T\nu - \mathbf{c})_i|$. This property allows us to reformulate the optimization problem in Equation 5 as follows:

$$\min_\nu \sum_{i=1}^{m} w_i |(A^T\nu - \mathbf{c})_i| \tag{6}$$

which can be rewritten as:

$$\min_\nu \|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1 \tag{7}$$

Notice that the resulting optimization problem is an *unconstrained* l_1 norm minimization where the decision variables correspond to the Lagrange multipliers $\nu \in \mathbb{R}^n$.

The form of the objective function is not surprising since the dual of the maxflow problem is the mincut problem which can be expressed as the following constrained l_1 norm minimization:

$$\min_{\xi \in \{0,1\}^n} ||\text{diag}(\mathbf{w})(A^T\xi - \mathbf{c})||_1 \tag{8}$$

[1] Remember that the edge weights $\mathbf{w}$ are non-negative

The difference between the objective function in Equation 8 and the one Equation 7 is the absence of constraints on ν . It is possible to show that the ν_i variables will converge to binary values without any external prodding. This can be seen by observing that the Lagrangian variables, $\lambda_{+_i}, \lambda_{-_i}$, corresponding to the edges that are *not* saturated by the max flow will converge to zero which in turn implies that the ν values associated with the endpoints of those edges will converge to the same value. This property allows us to conclude that the nodes will bifurcate into two equivalence classes, one involving s and one involving t, and that the ν values associated with these two classes will be 1 and 0 respectively. In other words, the constraints on the node label values, $\xi \in \{0, 1\}^n$, are superfluous and can be safely ignored which significantly simplifies the resulting optimization scheme.

The unconstrained formulation in Equation 7 is advantageous in many ways. It underlines the connection between graph cuts and convex optimization and allows us to employ continuous optimization techniques that can exploit the structure of the problem.

3. Implementation

The resulting unconstrained l_1 norm minimization problem described in equation 7 can itself be formulated as a linear program by introducing an auxiliary variable $\mathbf{y} \in \mathbb{R}^m$ where $\mathbf{y} \geq (A^T\nu - \mathbf{c})$ and $\mathbf{y} \geq -(A^T\nu - \mathbf{c})$ as described in [2]. The associated linear program is shown below.

$$\begin{aligned} & \min \mathbf{w}^T\mathbf{y} \\ \text{st} \quad & \begin{bmatrix} A^T & -I \\ -A^T & -I \end{bmatrix} \begin{bmatrix} \nu \\ \mathbf{y} \end{bmatrix} \leq \begin{bmatrix} \mathbf{c} \\ -\mathbf{c} \end{bmatrix}. \end{aligned} \tag{9}$$

This problem can be solved using the interior point method with logarithmic barrier potentials. In this approach the original linear program is replaced with the following convex objective function

$$\phi(\nu, \mathbf{y}) = t(\mathbf{w}^T\mathbf{y}) - \sum_{i=1}^{m} \log(\mathbf{y}_i - \mathbf{z_i}) - \sum_{i=1}^{m} \log(\mathbf{y}_i + \mathbf{z_i}) \tag{10}$$

Where $\mathbf{z} = (A^T\nu - \mathbf{c})$. The scalar t is used to weight the original objective function against the barrier potentials associated with the linear constraints.

This objective function is minimized using Newton's method. On each iteration of this procedure a locally optimal step, $[\Delta\nu\ \Delta\mathbf{y}]^T$, is computed by solving the following linear system:

$$\begin{bmatrix} A^T D_+ A & AD_- \\ D_- A^T & D_+ \end{bmatrix} \begin{bmatrix} \Delta\nu \\ \Delta\mathbf{y} \end{bmatrix} = - \begin{bmatrix} A(\mathbf{d}_1 - \mathbf{d}_2) \\ t\mathbf{w} - (\mathbf{d}_1 + \mathbf{d}_2) \end{bmatrix} \tag{11}$$

Where $\mathbf{d}_{1i} = 1/(\mathbf{y}_i - \mathbf{z}_i)$, $\mathbf{d}_{2i} = 1/(\mathbf{y}_i + \mathbf{z}_i)$; D_+ and D_- are diagonal matrices whose diagonal entries are computed as follows $D_{+ii} = (\mathbf{d}_{1_i}^2 + \mathbf{d}_{2_i}^2)$ and $D_{-ii} = (\mathbf{d}_{2_i}^2 - \mathbf{d}_{1_i}^2)$. By applying block elimination to factor out $\Delta\mathbf{y}$ the system can be further reduced to:

$$(A\text{diag}(\mathbf{d})A^T)\Delta\nu = -A\mathbf{g} \tag{12}$$

Where:

$$\mathbf{d}_i = 2/(\mathbf{y}_i^2 + \mathbf{z}_i^2) \tag{13}$$

and

$$\mathbf{g}_i = \frac{2\mathbf{z}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2} + \frac{2\mathbf{y}_i\mathbf{z}_i}{\mathbf{y}_i^2 + \mathbf{z}_i^2}(t\mathbf{w}_i - \frac{2\mathbf{y}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2}) \tag{14}$$

Once $\Delta\nu$ has been obtained from Equation 12, the $\Delta\mathbf{y}$ component of the step can be computed using the following expression.

$$\Delta\mathbf{y} = D_+^{-1}((\mathbf{d}_1 + \mathbf{d}_2) - t\mathbf{w} - D_-A^T\Delta\nu) \tag{15}$$

The entire interior point optimization procedure is outlined in pseudo-code as Algorithm 1. The input to this procedure is the vector of edge weights, $\mathbf{w}$.

Algorithm 1 Calculate min-cut: $\min_\nu \|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1$

```
choose t,μ
ν = 0.5
choose y such that y ≥ |A^T ν − c|
while change in l1 norm since last (outer) iteration above threshold1 do
    while change in l1 norm since last (inner) iteration above threshold2 do
        Compute d from Equation 13
        Compute g from Equation 14
        Solve (A diag(d) A^T)Δν = −Ag to get Δν
        Compute Δy from Equation 15
        If necessary, scale step by β so that ν + βΔν, y + βΔy are feasible.
    end while
    t = μ * t
end while
```

Note that the principal step in this procedure is the solution of the sparse linear system given in Equation 12 which means that the original l_1 norm minimization problem has been reduced to the problem of solving a sequence of sparse linear systems.

At this point we note that the matrix $L = (A\text{diag}(\mathbf{d})A^T)$ corresponds to a weighted graph Laplacian where the vector $\mathbf{d}$ indicates the weights that are to be associated with each of the edges of the graph. [2] The matrix is symmetric by construction and, since the entries in $\mathbf{d}$ are all positive, it is also positive definite. The entries along the diagonal of this matrix L_{ii} correspond to the sum of the weights of the edges impinging on the corresponding interior node in the graph - including the links to the s and t nodes. For the off diagonal elements, L_{ij}, it can be shown that $-L_{ij}$ will correspond to the weight of the edge connecting nodes i and j. This value will be zero if the two nodes are not connected.

From these two observations we can conclude that the matrix will be strictly diagonally dominant - that is the magnitude of the diagonal element will be greater than the sum of the magnitudes of the off diagonal elements since the diagonal entries will include the weights associated with the links to the s and t nodes which do not make an appearance in any of the off-diagonal entries.

The resulting sparse, banded matrix reflects the topology of the underlying grid. Matrices with this structure are frequently encountered in the process of solving partial differential equations, such as Poisson's equation, on two dimensional domains.

The numerical properties of the matrix L make the resulting linear system amenable to solution by the method of conjugate gradients [9]. Iterative techniques are preferred over direct techniques like Cholesky decomposition in this case because of the size of the matrix and the storage that would be required for the resulting factors. Pseudocode for the conjugate gradients method is presented in Algorithm 2.

Algorithm 2 Solve $A\mathbf{x} = \mathbf{b}$ using Conjugate Gradients

```
x = x0
r = b − Ax0
p = r
ρ = r^T r
while √ρ/||b|| > ε do
    a = Ap
    α = ρ/(a^T p)
    x = x + αp
    r = r − αa
    ρnew = r^T r
    p = r + (ρnew/ρ)p
    ρ = ρnew
end while
```

A distinct advantage of the conjugate gradient technique is that the steps in this algorithm can be readily parallelized. Each conjugate gradient step involves one matrix vector multiplication, 2 inner products and 3 SAXPY operations. All of these operations are amenable to implementation on the parallel architectures found on modern GPUs and multicore processors [16, 1].

[2] In fact the matrix L corresponds to the Graph Laplacian where the rows and columns associated with the s and t nodes are elided.

For the linear system given in Equation 12 we can exploit the fact that the matrix that we seek to invert has a regular structure derived from the underlying grid which further simplifies the matrix vector multiplication operation required on each iteration of the conjugate gradient procedure.

The conjugate gradient algorithm can be accelerated by choosing an appropriate symmetric preconditioning matrix, C, and then applying conjugate gradients to solve the system $(CAC)(C^{-1}\mathbf{x}) = C\mathbf{b}$ as described in [9]. The goal here is to choose a matrix C in such a way that the preconditioned matrix $CAC \approx I + B$ where B is a low rank matrix.

Concus, Golub and Meurant [6] describe preconditioning strategies that are specifically designed for the types of matrices that we seek to invert. Section 4 presents results that illustrate how effective these strategies can be in improving the convergence rate of the solver. However, the strategies described in [6] involve the solution of a series of tridiagonal matrices and these steps cannot be readily parallelized.

Experiments were also carried out using a simpler preconditioning strategy where the matrix C is chosen as follows $C = \text{diag}(\mathbf{a})$, $\mathbf{a}_i = 1/\sqrt{A_{ii}}$. In the sequel, we will refer to this as the diagonal preconditioner. When this preconditioner is applied to a diagonally dominant matrix, such as L, it produces a normalized variant where the diagonal elements are all 1 and the off diagonal elements all have magnitudes less than 1. Multiplying with such a diagonal preconditioner does not affect the fill pattern of the matrix.

4. Results

Experiments were carried out on graphs derived from actual image processing problems in order to determine how well the proposed scheme would work in practice. Since the scheme essentially reduces the mincut problem to the problem of solving a sequence of sparse linear systems, one can gauge the computational effort required to resolve a given graph cut by recording the total number of conjugate gradient iterations that are performed in the course of driving the system to convergence. Three variants of the scheme were used in these experiments, the first variant employed the conjugate gradient method without any preconditioning, the second made use of the diagonal preconditioner described in the previous section while the third used the preconditioning strategy described by Concus, Golub and Meurant [6].

The proposed scheme was applied to the image restoration problem described by Boykov *et al*. in [5]. On the example shown in Figure 2 the proposed scheme required 237 CG iterations with no preconditioner, 156 with the diagonal preconditioner and 32 with the Concus,Golub and Meurant preconditioner.

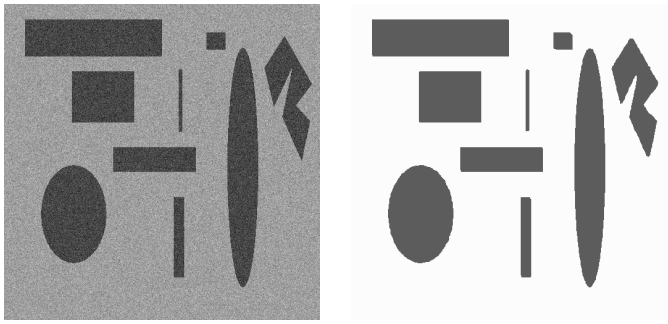

Figure 2. Example of image restoration using the proposed scheme.

Table 1. Number of iterations taken for separating the foreground in each image in fig 3, using different preconditioning strategies

Image name	**CG iters**	**with diagonal preconditioner**	**with preconditioner in [6]**
Ayers rock	393	107	25
Tomb	359	157	36
Liberty Bell	495	189	29
Footballer	441	183	39
Family	420	190	42
Superman	752	167	39
Actress	878	366	85
Giraffe	543	269	58

The method was also applied to the foreground/background segmentation problems shown in Figure 3. In these experiments the underlying weighted graphs were constructed using the GrabCut algorithm described by Rother, Kolmogorov and Blake [20]. All of the images in question are 512×512. Table 1 shows the number of conjugate gradient iterations taken by each of the three variants of the optimization procedure.

These results demonstrate that the preconditioning schemes are, in fact, quite successful at accelerating the convergence of the conjugate gradient procedure in this situation. The diagonal preconditioner reduces the number of iterations required by a factor of 0.4 on average while the Concus and Golub preconditioner reduces the complexity even further. Table 2 shows the total number of floating point operations required to solve each of the segmentation problems using the diagonal preconditioner.

Experiments were also carried out to gauge how the computational effort required to compute the segmentation varied with the size of the input image. Table 3 shows how the number of conjugate gradient iterations changed as the scheme was applied to scaled versions of a given image. The diagonal preconditioner was used in this set of experiments. These results provide some indication of how the number of conjugate gradient iterations required increases witht image size. The computational effort required to perform a single conjugate gradient iteration will increase lin-

Table 2. Total number of floating point operations needed for foreground extraction for images in Fig 3.

Image name	**Floating point operations** (x 10^9)
Ayers rock	1.37
Tomb	1.45
Liberty Bell	1.69
Footballer	1.67
Family	1.63
Superman	1.79
Actress	2.96
Giraffe	2.07

Table 3. Variation of the number of Conjugate Gradient iterations with image size

Image size	**CG iters**
128x128	107
256x256	145
512x512	192

Table 4. Time taken for a single Conjugate gradient iteration on images of different sizes

Image size	**CG iteration time(ms)**
128x128	0.4
256x256	0.6
512x512	1.6
1024x1024	6.2
2048x2048	29.7

early with the size of the image.

The proposed scheme was also implemented on an NVidia GeForce 7800 GTX GPU. In this implementation, the weights associated with the nodes and edges were stored as single precision floating point textures. Basic operations such as matrix vector multiplication, SAXPY operations and inner products were implemented as Cg pixel shaders. The conjugate gradient solver was implemented by combining these basic operations in the appropriate sequence.

The time taken to perform a single conjugate gradient iteration for images of different size is given in Table 4. As expected the computational effort increases linearly with the number of pixels in the image except for very small images where overhead costs seem to constitute a more significant fraction of the computation. This implementation employed the diagonal preconditioning procedure.

Each conjugate gradient iteration requires $18n$ floating point operations. So the effective delivered performance of the system is approximately 3.5 Gflops. At present the implementation appears to be memory bound since this particular GPU has a theoretical peak computational performance of around 35 Gflops [3]. The implementation was applied to the segmentation problems shown in Figure 3 and the timings achieved on these 512x512 problems are summarized in Table 5.

Table 5. Time taken to extract the foreground of images in Fig 3 on the GPU.

Image name	**Time taken (in secs)**
Ayers rock	0.60
Tomb	0.82
Liberty Bell	0.69
Footballer	0.71
Family	0.79
Superman	0.59
Actress	1.24
Giraffe	1.01

5. Conclusion

In this paper the graph cut problem is rephrased as an unconstrained l_1 norm minimization problem. In this formulation, the graph cut problem is viewed as a convex optimization problem defined in terms of a vector of node weights, $\nu \in \mathbb{R}^n$, which can be solved using the barrier method. The resulting l_1 norm minimization problem can be reduced to the problem of solving a sequence of sparse linear systems involving the graph Laplacian. This Laplacian matrix posseses a number of useful numerical properties which make it amenable to numerical solution using the method of conjugate gradients. We can also take advantage of the fact that linear systems of this form have been studied extensively in the context of partial differential equations on 2D domains and a number of effective techniques for solving them have been developed.

The fact that the node weights, ν, will converge to binary values at the global minimum is a useful property with important practical consequences. Firstly, it means that the optimization procedure yields the node labels immediately, without the need for an intervening flow interpretation. Secondly, the fact that the weights tend towards discrete values makes it easy to employ rounding as the barrier method approaches convergence. It also reduces the numerical precision required to execute the algorithm; in practice, one can carry out the procedure using *single-precision* floating point arithmetic. Contrast this with the problems one encounters in applying the barrier method to the max flow problem where numerical issues can make it difficult to determine whether a given link is saturated with flow or merely close to saturation.

Importantly, the entire procedure can be carried out us-

[3] Performance figures obtained from the Stanford Graphics Laboratories GPUBench benchmark suite.

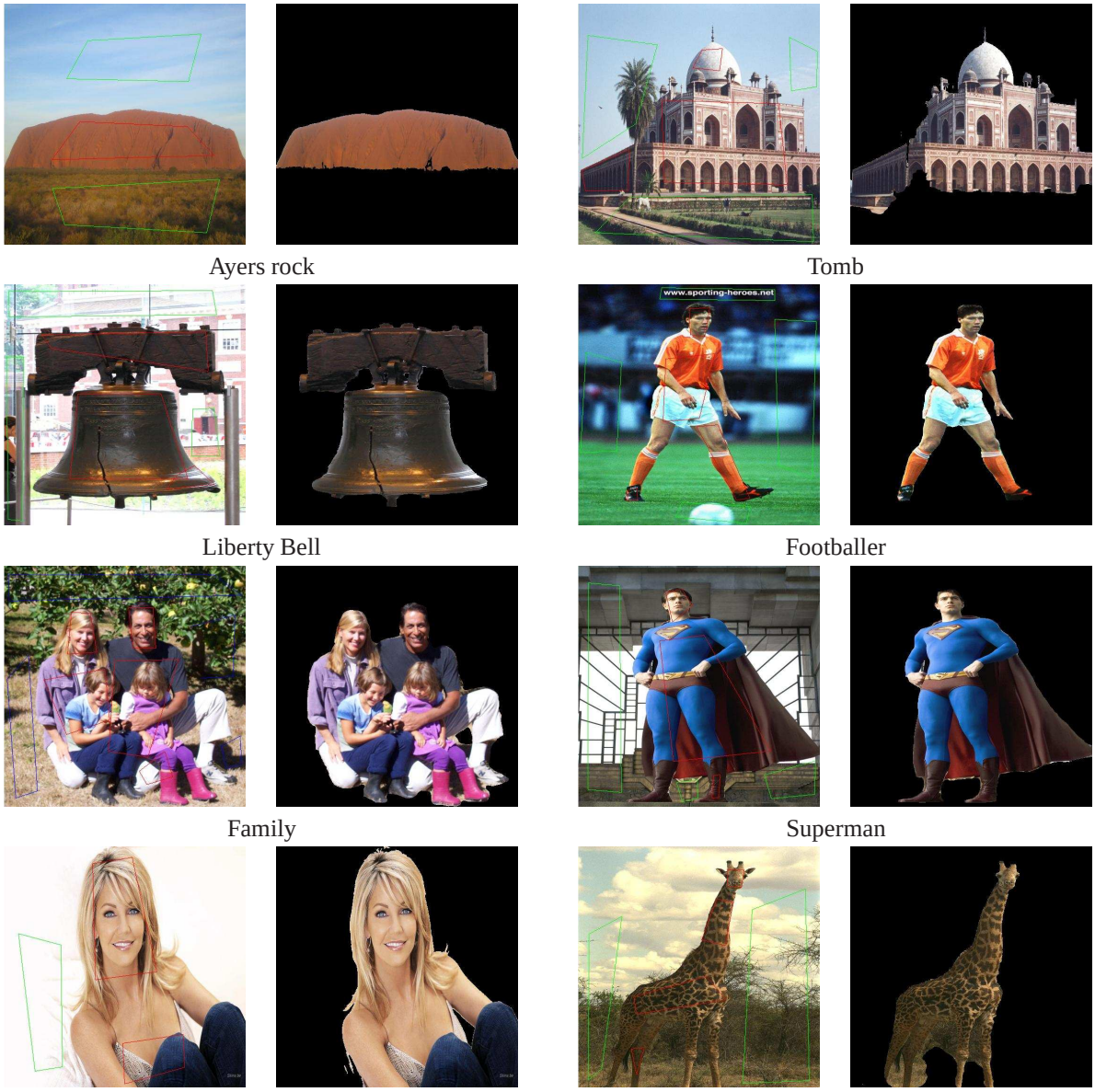

Figure 3. These segmentation examples were used to test the graph cut implementation described in the previous sections. The graph weights were computed using a variant of the Grab Cut procedure described in [20].

ing vector operations which are highly amenable to parallelization. This means that the system is well suited to implementation on modern multi-core CPUs and GPUs which offer an abundance of single precision floating point performance to applications that are structured to take advantage of the available parallelism. Consider the remarkable advances in CPU and GPU performance that have been enabled by decreasing feature size and increasing parallelism. Table 6 summarizes the single precision floating point performance offered by a number of current systems. High performance dual core processors are currently standard on desktops, quad-core and teraflop systems from Intel are on the horizon; GPU performance is increasing even faster. The proposed implementation scheme is designed to exploit the performance afforded by these types of systems.

6. Future Work

While this paper has discussed the graph cut problem using the basic grid topology where every pixel is connected to 4 of its neighbors, it is straightforward to extend the analysis to handle the situation where each pixel has links to all 8 of its neighbors or to the 3D grids encountered in medical

Table 6. Single Precision Floating Point performance afforded by a range of current CPUs and GPUs

Processor	GFLOPS
Intel Core 2 Duo	8
NVidia 7800	35
ATI R580	125
NVidia 8800	173
Sony/IBM Cell	205

imaging.

On another front, efforts are currently underway to further optimize the existing GPU implementation to make better use of the available floating point units. We also intend to port the implementation to the recently released NVidia 8800 GPU which offers a Unified Shader Architecture with greater performance and memory bandwidth.

Further work is needed to determine whether the linear systems encountered in the optimization procedure could be solved on the GPU using other numerical methods like the cyclic reduction technique employed by Kass, Lefohn and Owens [12].

Coarse to fine and multigrid approaches are also natural candidates for further investigation. The results obtained by solving the minimization problem on a coarser level could be used as a starting point for the optimization procedure on the finer scale, hopefully with a concomitant decrease in the number of iterations required to achieve convergence.

Reformulating the graph cut problem as a continuous optimization problem prompts us to consider whether a change in coordinates may further simplify the optimization problem. For example, it may be fruitful to investigate whether a wavelet transform of the original node weight vector would yield any advantage or insight.

References

[1] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. In *SIGGRAPH*, 2003. 4

[2] S. Boyd and L. VandenBerghe. *Convex Optimization*. Cambridge University Press, 2004. 3

[3] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, 2001. 1

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26, Sept 2004. 1

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 23(11), Nov 2001. 1, 5

[6] P. Concus, G. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM Journal of Scientific Computing*, 6(1), 1985. 5

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2002. 1

[8] N. Dixit, R. Keriven, and N. Paragios. Gpu-cuts: Combinatorial optimisation, graphic processing units and adaptive object extraction. Technical report, CERTIS,ENPC, March 2005. 1

[9] G. Golub and C. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996. 4, 5

[10] H. Ishikawa and D. Geiger. Occlusions, discontinuities and epipolar lines in stereo. In *ECCV*, 1998. 1

[11] O. Juan and Y. Boykov. Active graph cuts. In *CVPR*, 2006. 1

[12] M. Kass, A. Lefohn, and J. Owens. Interactive depth of field using simulated diffusion on a gpu. Technical report, Pixar Animation Studios, 2006. 8

[13] P. Kohli and P. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *ICCV*, 2005. 1

[14] V. Kolmogorov and R. Zabih. Computing visual correspondance with occlusions via graph cuts. In *ICCV*, 2001. 1

[15] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts. *PAMI*, 26(2), 2004. 1, 2

[16] J. Kruger and R. Westermann. Linear algebra operators for gpu implementation of numerical algorithms. In *SIGGRAPH*, 2003. 4

[17] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *ICCV*, 2005. 2

[18] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Prentice-Hall, 1982. 2

[19] S. Paris, F. Sillon, and L. Quan. A surface reconstruction method using global graph cut optimization. *IJCV*, 66(2), 2006. 1

[20] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. In *SIGGRAPH*, 2004. 1, 5, 7

[21] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *ICCV*, 1998. 1

[22] S. Sinha and M. Pollefeys. Multi-view reconstruction using photo-consistency and exact silhoutte constraints: A maximum flow formulation. In *ICCV*, 2005. 1

[23] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *CVPR*, 2000. 1

[24] B. Taskar. *Learning Structured Prediction Models: A Large Margin Approach*. PhD thesis, Stanford University, 2004.