APPLYING CONVEX OPTIMIZATION TECHNIQUES TO ENERGY MINIMIZATION PROBLEMS IN COMPUTER VISION

Arvind Bhusnurmath

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2008

C.J.Taylor
Supervisor of Dissertation

Rajeev Alur
Graduate Group Chairperson

Acknowledgements

Acknowledgements will appear in the final dissertation document.

ABSTRACT

APPLYING CONVEX OPTIMIZATION TECHNIQUES TO ENERGY MINIMIZATION PROBLEMS IN COMPUTER VISION

Arvind Bhusnurmath

C.J.Taylor

Energy minimization is an important technique in computer vision that has been applied to practically all early vision problems. Several methods have been proposed to solve the problem and to date, the most popular ones have been discrete optimization approaches such as graph cuts and belief propagation. While convex programming approaches have been proposed, the resulting problems were practically unsolvable due to their size. This thesis describes a novel approach to applying convex optimization, in particular interior point barrier methods, to solve some energy minimization problems in an efficient manner. The methods exploit the fact that computer vision problems usually have specific structure to them.

Two problems are handled in this thesis. In the first, a popular energy minimization technique, graph cuts, is reduced to an unconstrained ℓ_1 norm minimization problem. This problem is then solved using linear programming. The resultant algorithm is parallelizable and the thesis provides one possible implementation using a graphical processing unit. Results of using this approach on interactive foreground-background segmentation problems are provided.

In the second problem, the energy function associated with stereo matching is directly converted into a convex function via an approximation of the data penalties. The resultant linear program is again solved using interior point methods and exhibits very similar structure to the first approach. Results on the standard Middlebury stereo matching dataset will be discussed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A wide variety of problems in computer vision can be conceptualized as labeling problems where quantities such as disparities in the case of stereo matching, segment numbers in the case of segmentation and velocities in the case of motion have to be assigned to every pixel in an image. An extremely popular approach to solving such problems is to recast them as energy minimization problems. The goal here is to formulate an energy function such that the energy minimum is attained at the pixel labeling that represents the best solution to the problem. This thesis provides a framework for solving a class of energy minimization problems that are relevant to computer vision using ideas from the richly developed field of convex programming.

The basic solution mechanism is as follows. Given an energy minimization problem, the first step is to formulate it as a linear program such that the solution to the linear program corresponds to a minimum energy configuration. These linear programs can be solved using the barrier method in which the linear inequalities are approximated using logarithmic barrier potentials. The barrier method is an iterative technique that improves the optimization solution through successive applications of Newton's method. The major computational effort in applying Newton's method is the solution of a system of linear equations involving the gradient and Hessian of the resulting objective function.

An interesting aspect of the computer vision problems considered in this thesis is that the linear equations they give rise to are sparse and highly structured and hence amenable to efficient numerical methods. This is a crucial aspect of this work since the linear programs being dealt with are considerably large.

This thesis develops a general solution framework which minimizes certain energy functions via convex optimization. The first step is to convert the energy minimization into a linear program. The linear program is then solved using the interior point barrier method. Within the barrier method, several Newton steps need to be taken and these steps are solved for via a system of linear equations. The thesis shows how the structure of the problem gets reflected in these linear equation systems.

This solution framework is applied to two different computer vision problems. Firstly, the popular graph cuts technique of energy minimization is converted into an unconstrained ℓ_1 norm problem. This problem is solved via linear programming. The method is used in this thesis to solve interactive foreground-background segmentation problems such as the one shown in Figure 1.1. The solution proposed in this thesis is highly parallelizable and an implementation of the algorithm on a GPU is described in Chapter 6.

The other problem being solved in the thesis is that of stereo matching. An example is shown in Figure 1.2. Unlike segmentation wherein pixels have to be labeled with discrete values like foreground or background, pixels in stereo can take on any scalar value. This aspect comes out in a more natural manner when the linear programming formulation is used. It enables subpixel values to be directly incorporated during the optimization stage as opposed to a post processing step.

The rest of this chapter is divided into the following sections. Section 1.1 discusses some prior work regarding the various computer vision domains where energy minimization has been used. Common methods used to find the minima of energy functions are surveyed in 1.2. Section 1.3 lists the major contributions of this research. Section 1.4 lays out the outline for the remainder of the thesis.

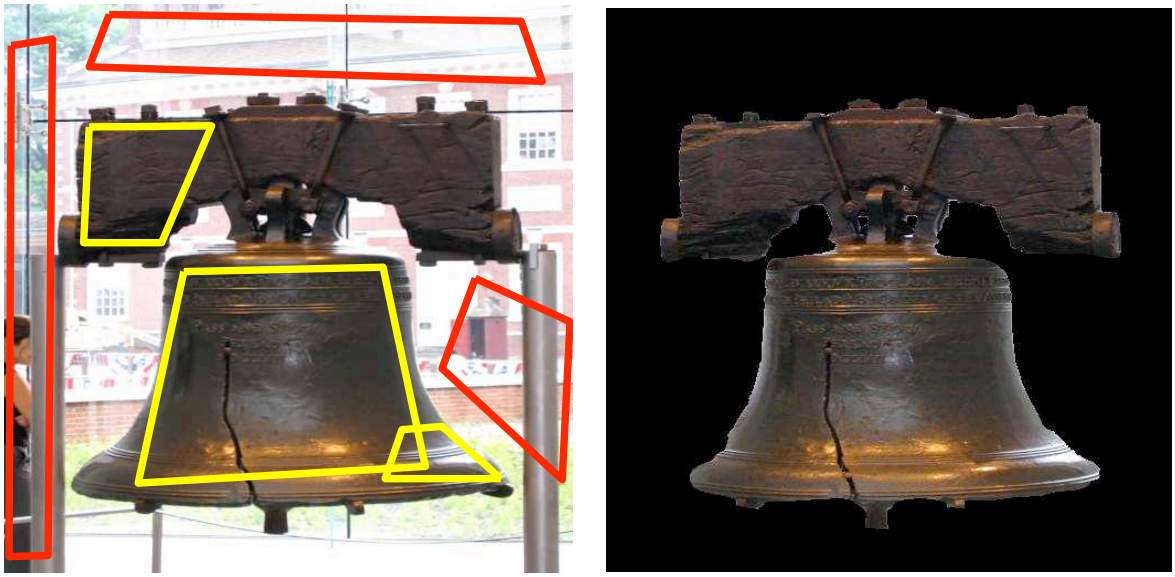


Figure 1.1: An example of the foreground-background problem. The user marks out regions of foreground and background as shown in the left image. The extracted foreground object is shown on the right.

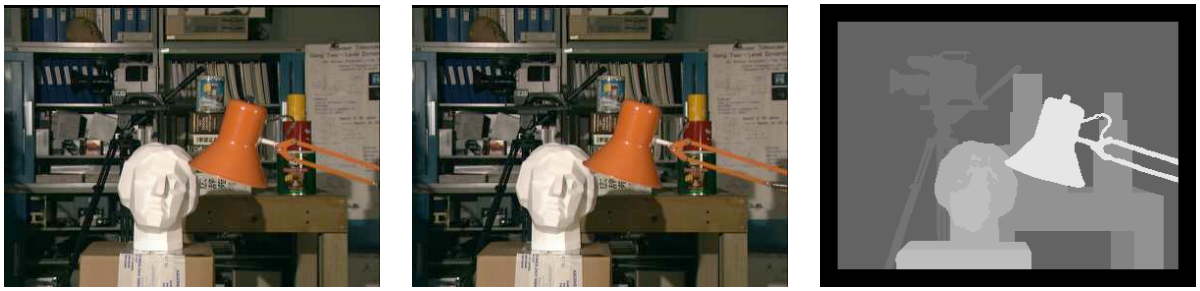


Figure 1.2: An example of the stereo matching problem. The first two images from the left constitute the pair of images whose pixels need to be matched, and the image on the right is the ground truth disparity map.

1.1 Energy minimization problems in computer vision

The energy minimization framework essentially tries to incorporate knowledge from the given data and prior knowledge about the nature of the solution in order to find the best possible labeling for a problem. The usual prior that is assumed is that of smoothness - neighbouring pixels typically have similar labels. To incorporate this assumption into the energy framework, a neighbourhood system on the pixels is defined. A clique is defined as a subset of pixels such that any two distinct pixels are mutual neighbours. A set of clique potentials that specify energy values depending upon the labels being assigned to pixels that form the clique is specified. The energy function is then formulated in terms of a Gibbs energy [13] which is composed by summing these clique potentials. It is important to note that Gibbs energy formulations and Markov random field(MRF) formulations are equivalent due to the Hammersely-Clifford theorem [40]. Therefore an equivalent method of looking at the process of finding the minimum energy is that of finding the maximum a posteriori(MAP) estimate of the MRF. Geman and Geman [32] have a detailed explanation of this formulation in their landmark paper. Li [67] provides more details on how MRF models have been used in the field of computer vision.

The general form of an energy function (notation taken from work by Boykov, Veksler and Zabih [12]) is as follows

$$E(L) = \sum_{p \in P} D_p(L_p) + \sum_{p,q \in N} V_{p,q}(L_p, L_q), \tag{1.1}$$

where $L = \{L_p | p \in P\}$ is the current labelling of the pixels, D_p is called the data penalty function and indicates the compatibility of the labelling with the given data, $V_{p,q}$ is called the interaction potential or the smoothness function and N is the set of all pairs of neighbouring pixels. The smoothness term incorporates the notion of a piecewise smooth world and penalizes assignments that label neighboring nodes

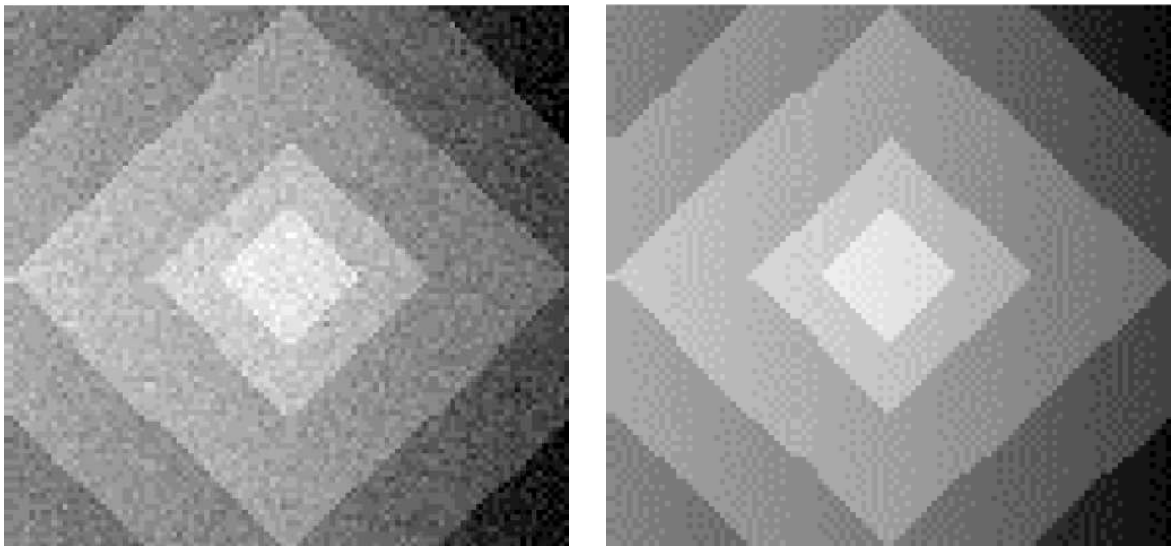


Figure 1.3: Example of image restoration taken from [12]. The figure on the left is the noise-corrupted image and the one on the right is the original image that we desire to recover.

differently.

Let us look at an example computer vision problem that has been solved using energy minimization. Consider the problem of image restoration and the solution proposed in [12]. An image is corrupted with noise and the aim is to recover the original image as shown in Figure 1.3.

In this case the labels to be assigned are the 'true' intensity values of the pixels. Boykov *et al* [12] use a data term

$$D_p(L_p) = \min(|L_p - I_p|^2, M), \tag{1.2}$$

where L_p is the restored intensity label and I_p is the observed intensity. M is a constant that ensures robustness to outliers. This says that the restored intensity label should be close to the currently observed intensity. The smoothness term being used is

$$V(L_p, L_q) = k * \min(k1, |L_p - L_q|), \tag{1.3}$$

which ensures two properties. Firstly a pair of neighbouring pixels that have differing intensity labels will be penalized. This ensures smoothness. However we want to break the smoothness assumption at places where we think there is an edge. An edge would be characterized by a fairly large difference between the neighbouring intensity values. Therefore the penalty is thresholded by $k1$ so that we do not get a denoising result where there is bleeding across edges. A multiplicative constant k is also introduced to balance the contribution of the smoothness terms versus the data terms.

In a manner very similar to this, a number of vision problems have been treated using energy minimization. Ishikawa and Geiger [47] and Roy and Cox [79] both use a graph cut formulation to minimize energy for the stereo correspondence problem although the construction is different from the one we will discuss in section 2.1. Both these formulations aim to label each pixel with its corresponding disparity value. Kolmogorov and Zabih [57] were able to achieve very good results on stereo problems as is evidenced by their excellent performance on the Middlebury dataset [80, 81] for stereo. In their formulation labels are assigned to pairs of pixels. The label either indicates that the pixels are matched or that they are unmatched. Just like stereo, energy minimization can be used for motion [12], except now the data-term will involve computation in two dimensions as pixels are displaced in both directions. In both stereo and motion a standard technique to ensure the data term is insensitive to image sampling is to interpolate between discrete pixel values as described in [6].

In the field of segmentation the problems that have been tackled include interactive foreground background [10, 78] and depth based image segmentation [70].

Energy functions defined on segments or layers as opposed to just pixels have worked successfully too. Examples include Hong and Chen [45]'s work on stereo, Wang and Adelson's work on motion [93] and Criminisi *et al* .[22]'s work on video segmentation.

The approach can be applied for 3D problems as well. Sinha and Pollefeys [83]

reconstruct the surface of an object using multiple calibrated images and silhouette. The energy function incorporates both photo consistency constraints and silhouette constraints. The voxel occupancy problem is dealt with in Snow *et al* [85]. As opposed to labeling pixels, in 3D the problem turns into one of labeling voxels.

As we can see from all these examples, while the general form of the energy function remains the same, success in solving the problem depends on careful selection of the data term and smoothness terms. The solution methodology is widespread enough to now merit a conference of its own - EMMCVPR (Energy Minimization Methods in Computer Vision and Pattern Recognition).

1.2 Methods of finding the minimum energy

Energy minimization problems in vision have been solved by a variety of techniques. A comparative study of these methods can be found in Szeliski *et al* [88]. All the approaches have an underlying notion of a graph that is defined typically with pixels as the nodes, and edges that represent how one pixel's label affects its neighbours.

While in most cases finding the exact energy minimum is NP-hard, there are some specific situations in which this can be achieved and those are discussed first.

Ishikawa [46] shows that an exact minimum can be found if the smoothness function is convex in terms of a linearly ordered label set. The method maps the problem into that of finding a minimum cut in a directed graph. The graph is made up of columns corresponding to each pixel. Within each column there exist vertices corresponding to the assignment of different labels to the pixel. The graph is constructed such that there is exactly one edge in the cut from each column. The tail of this edge then identifies the label that needs to be assigned to the pixel. The convexity criterion causes issues since it is not robust around discontinuities and hence has difficulties preserving edges. The other problem is that the size of the graph gets large as the number of labels increases since each potential (pixel,label)

assignment has to be represented by a node in the graph. Finally, this method only deals with pairwise clique potentials.

The other situation where an exact minimum can be found is when the labeling is binary. Under binary labeling and with clique potentials up to size 3, as long as the so called regularity condition is satisfied [58], the exact minimum can be computed using graph cuts. A more detailed discussion of graph cuts is presented in Chapter 2.

The two cases detailed above do not cover a lot of situations that arise in energy minimizations. For example the simple Potts model for a multiple label setting is not covered. The Potts energy function is described as

$$V(L_p, L_q) = \begin{cases} 0 & \text{if } L_p = L_q \\ \gamma & \text{otherwise.} \end{cases} \tag{1.4}$$

which basically implies all discontinuities get penalized uniformly.

In such situations the problem becomes NP-hard and then the best that one can hope for is a good approximation algorithm. The most popular methods for solving these problems are graph cuts (using swap moves or expansion moves) or belief propagation.

Graph cuts are covered in Chapter 2 but for the sake of comparison purposes in this section a few points need to be made. Graph cuts are able to efficiently minimize a certain subset of energy minimization problems where the *regularity* condition is satisfied. Convergence is not an issue in this method of solution. In cases where the smoothness function is a metric function, the solution has an energy that is at most twice the minimal value.

Max-product belief propagation can be conceptualized as a message passing framework. Let m_{pq}^t be the message being sent from p to its neighbour q at time t. Initializing all the initial messages to 0, one computes the new messages using the following equation [26].

$$m_{pq}^{t}(f_q) = \min_{f_p} \left(V(f_p, f_q) + D_p(f_p) + \sum_{s \in N(p)-q} m_{sp}^{t-1}(f_p) \right) \tag{1.5}$$

where f_p and f_q are the labels that are being assigned to pixels p and q respectively and $N(p)$ denotes the neighborhood of p.

At the end of T iterations the label that gets chosen for pixel q is the one that minimizes the belief that is given by

$$b_q(f_q) = D_q(f_q) + \sum_{p \in N(q)} m_{pq}^{T}(f_q). \tag{1.6}$$

Felzenszwalb and Huttenlocher [26] discuss methods of speeding up belief propagation for typical vision problems. Their implementation has been used to solve most energies that are minimized via belief propagation.

The interesting aspect of belief propagation is that it makes virtually no assumptions about the nature of the energy function. However because of its message passing nature, it is easy to see the potential problems that can be caused by loops in the graph. While convergence is not guaranteed it is interesting to note that this scheme has been used quite successfully to solve several energy minimization problems. Empirical work by Tappen and Freeman [89] indicates that in cases where belief propagation and graph cuts can both be applied, belief propagation converges to a higher energy value. However, in terms of overall quality, belief propagation does not seem to suffer in comparison to graph cuts.

Weiss and Freeman [94] showed that in an arbitrary graph the max product assignment is a local optimum over any subset of nodes that form a node induced subgraph with at most one cycle per connected component. Wainwright [92] extends this to show that the max-product assignment is optimal over any subgraph that contains at most one cycle per connected component.

While the main concern while using a belief propagation method is convergence, as an approximation algorithm it is also worthwhile to consider what energy value it

converges to or alternatively how much the labeling it converges to differs from the MAP labeling. As per [92] the max product labeling either agrees with the MAP labeling or differs from this labeling by a Hamming distance larger that the following function

$$l(g) = 2g - \lfloor g/2 \rfloor - 1 \tag{1.7}$$

where g is the girth of the graph, which is the number of edges in the shortest cycle. This error bound is quite different from the constant factor approximation that graph cuts provides, however the energy functions being tackled are far more general. In particular, for 4 connected grid graphs that are very common in computer vision application this result shows that the max product labeling differs from the MAP labeling on at least 5 pixels.

A variant of the belief propagation technique which is rapidly gaining ground is tree-reweighted message passing or TRW [91] which tries to maximize the lower bound of the energy function by a convex combination of marginal values that are obtained on trees embedded within the graph by performing belief propagation only on these trees. This work is improved upon in Kolmogorov's work in [56] wherein the message passing is done in a specific sequential manner (hence giving it the name TRW-S) to guarantee convergence. In [56] the updates are performed in a sequential manner that ensures that the value of the lower bound (that is being maximized) never decreases. This still does not guarantee that the algorithm converges to the global maximum. Rather it converges to a maximum with respect to the TRW algorithms. This method also uses half as much memory as traditional message passing approaches.

Linear programming techniques for labeling problems were initially proposed by Kleinberg and Tardos [53] and then by Chekuri *et al* [17] but due to the size of the resulting problems, were not actively used by computer vision researchers.

Consider a labeling problem with vertex set V, edge set E, label set L and

a smoothness function given by $V_{pq}(L_p, L_q) = w_{pq} d_{L_p, L_q}$ where d_{L_p, L_q} defines the penalty for assigning labels L_p to p and L_q to q and this gets weighted by w_{pq}. Then as per [17] the energy minimization problem can be formulated as an integer optimization problem as follows

$$\min \quad \sum_{p \in V, a \in L} D_p(a) x_{p,a} + \sum_{(p,q) \in E} w_{pq} \sum_{a,b \in L} d_{ab} x_{pq,ab} \tag{1.8}$$

$$\text{st} \quad \sum_{a} x_{p,a} = 1 \qquad \forall p \in V \tag{1.9}$$

$$\sum_{a} x_{pq,ab} = x_{q,b} \qquad \forall b \in L, (p,q) \in E \tag{1.10}$$

$$\sum_{b} x_{pq,ab} = x_{p,a} \qquad \forall a \in L, (p,q) \in E \tag{1.11}$$

where $x_{p,a}$ is a 0-1 variable that is indicative of the pixel p being labeled a. The 0-1 variable $x_{pq,ab}$ indicates the pixel p is labeled a and pixel q is labeled b. This optimization problem is an integer optimization problem which is in the class of NP-hard problems. To solve it approximately, one of the methods is to relax the 0-1 constraints to $x_{p,a} \geq 0$ and $x_{pq,ab} \geq 0$.

Current work by Komodakis and Tziritas [59] which gets further refined in [60] utilizes duality theory on this relaxed formulation. They propose a fast algorithm called Fast-PD, using the primal-dual method of convex optimization. The goal of this method is to continually update both primal and dual solutions such that eventually one obtains a pair of primal and dual solutions that are both feasible and the ratio of the primal objective to the dual objective is small. If this ratio is given by f_{app} then Fast-PD provides an f_{app} solution to the integral optimization problem. They also provide a connection to the α expansion graph cut algorithms. In fact in a manner similar to graph cuts, each inner iteration of the algorithm involves computing the maximum flow in an appropriately defined flow network. The place where this algorithm gains its mileage is that as the algorithm proceeds, the number of nodes that are connected to the source gets reduced. As a result, there are fewer

augmenting paths and hence faster computation of maximum flow. For example, while solving the Tsukuba stereo problem [60] is able to obtain a speed up by a factor of 5.

There are some assumptions that need to be made in order for Fast-PD to work. It requires that $d(a,b) \geq 0$ and $d(a,b) = 0 \iff a = b$. Under these assumptions, when one uses Fast-PD the factor of approximation is $2\ \frac{d_{max}}{d_{min}}$ where d_{max} and d_{min} denotes the maximum and minimum penalty for assigning different labels to connected pixels.

Overall, Linear Programming(LP) has not found much success yet in the domain of energy minimization even though the TRW methods are essentially tackling the dual of the relaxed LP proposed in Chekuri *et al* . Yanover *et al* [97] perform an empirical evaluation of linear programming techniques versus belief propagation and show how TRWs ability to exploit the structure of the problem allows it to solve problems that cannot be solved even by the best of the commercial linear programming softwares [1, 2]. The main problem with LP is the size of the problem.

Apart from the linear programming relaxation, it is also possible to convexify the integer programming problem by using different kinds of convex relaxations - Quadratic Programming(QP) [76], Semi-Definite Programming [90] and Second Order Cone Programming(SOCP) [64]. It is also handled using a spectral relaxation in Cour and Shi [21]. In Kumar *et al* [63] these different relaxations are compared and it is proven that the linear programming relaxation provides a better approximation than the others. However, none of these methods handle large scale problems effectively. They handle a more general class of energy functions than graph cuts and as a result of being convex programs, convergence is guaranteed. In the sequel, by treating energy minimization from a continuous optimization perspective, we will see how certain large but structured problems can be solved efficiently using linear programming.

A method of energy minimization that has a great deal of similarity with the graph cuts approach in this thesis is the Random Walker image segmentation by

Grady and Funka-Lea [36] which starts off with a few labeled pixels and then determines the probability that a random walker starting at the unlabeled pixels will first reach one of the labeled pixels. The biases of the random walker to visit the vertices are related to the smoothness function. Connections are shown between this method and the discrete Dirichlet problem from PDEs and the method of solution involves solving a system of equations involving the Laplacian matrix [72]. Grady *et al* [37] use this method to perform seeded segmentation on a GPU. Their approach has mainly been applied to 3D data that is obtained from the field of medical imaging.

The approach taken in this thesis to solve the graph cuts problem also eventually ends up solving a sparse system of equations involving a graph Laplacian and has connections with the field of PDEs. Viewing the random walker algorithm from the point of view of energy minimization, in Sinop and Grady [84], it is shown that both graph cuts and random walker minimize the same energy except with different norms. The graph cuts algorithm uses ℓ_1 while the random walker uses ℓ_2. Hence, the random walker algorithm can potentially be treated as an alternative energy minimizer in situations where the graph cuts algorithm has been applied.

In this thesis, the connections between graph cuts and ℓ_1 norm minimization have been independently proven. Also efficient ways of performing the ℓ_1 norm minimization in a manner such that it exploits the structure of the underlying graph are explored.

So far we have only looked at minimizing energy functions that involve clique potentials of sizes at most 2. Adding in clique potentials for cliques of greater sizes should be advantegous in modeling the problem. However, the usage of such energy functions has been limited by the lack of efficient algorithms for minimization. Recent work by Kohli *et al* [55] and Lan *et al* [65] is able to make some headway in the solution of such problems. Kohli *et al* manage to provide some characterization of energy functions involving higher order terms that can be minimized by polynomial time algorithms (this is a result of submodularity. More details can be found in [24]).

Also in the cases where the higher order terms are generalized potts model energies

$$\psi_c(x_c) = \begin{cases} \gamma_k & \text{if} \quad x_i = l_k, \forall i \in c, \\ \gamma_{max} & \text{otherwise.} \end{cases} \tag{1.12}$$

,

the energy can be minimized by finding a graph cut and hence for such energies the minimization is much faster. In their work, Kohli *et al* apply this method to the problem of texture based segmentation. Lan *et al* use 2x2 clique potentials and minimize the resulting energy using belief propagation for the purposes of image denoising. Methods are provided in their work for reducing the computational cost of performing belief propagation on higher order cliques by reducing the set of labels being explored in each clique. The results obtained for image denoising are considerably better than those obtained using pairwise clique potentials. However the time taken to minimize the energy is much greater.

1.3 Contributions of the thesis

The main contribution of this thesis is that it provides a continuous optimization method to solve certain global energy minimization problems. Almost all successful methods that minimize energy come from the field of discrete optimization. However certain problems in computer vision such as stereo and motion involve inherently continuous variables and therefore the methods proposed in this thesis should be more applicable in those circumstances.

Even though the optimization is viewed from a different perspective, the problems are still very large in size. Therefore, in terms of implementation, the thesis makes contributions in terms of methods of exploiting the structure in the problems being addressed. It is important to note that the linear programs in this thesis were formulated and input into commercial solvers such as TOMLAB [2] and MOSEK [1] but neither software was able to solve them since these generic solvers were unable

to identify the problem structure.

The linear programs ultimately get reduced to solving systems of sparse and highly structured linear equations. These systems have been well studied in other areas, such as the solution of partial differential equations using finite difference methods, and this opens up the prospect of porting ideas from that domain.

There are also specific contributions being made to the two problems that are addressed within this thesis.

The main contributions of the method for solving graph cuts are as follows

- An unconstrained ℓ_1 norm formulation of the graph cuts problem is developed whose solution, in our opinion, provides a more intuitive understanding of pixel labeling than the flow based approaches. Further we are able to draw connections to existing work on ℓ_1 norm minimization in the the optimization community.

- The system of equations that arise out of the Newton step in this problem are similar to the Poisson PDEs. The matrix is block tri-diagonal and good preconditioning strategies exist for such matrices.

- A parallelizable algorithm for solving the system of equations is provided that is implementable on a host of architectures. A particular implementation on the NVIDIA 8800 GPU is described in this thesis.

The contributions of the new stereo matching method are as follows:

- A new formulation of the stereo matching problem in terms of linear programming is provided. Minimization of the energy function is done via a convex objective function which approximates the original energy function. The well developed theory of convex programming can then be applied.

- All possible disparity values are considered during any step of the process. This is in contrast to an α expansion framework wherein at each stage the only

possible change is that some pixels change their labels from their currently assigned label to the label α.

- Subpixel disparities can be directly incorporated into the optimization framework right at the beginning since we are solving the problem using continuous optimization. Usually subpixel computations are done as a post processing step and we feel that integrating this into the optimization process will give better results.

- The ability to incorporate a penalty term in the objective function that involves the Laplacian of disparity. While the Laplacian involves interaction between 3 pixels, most conventional methods restrict their energy terms to only pairwise clique potentials. This extra term should account for foreshortening effects in a better manner.

1.4 Document Layout

This thesis is divided into two parts. Chapters 2 through 7 describe a linear programming formulation of graph cuts to solve energy minimization problems. In particular interactive foreground-background segmentation is solved using this methodology. Chapters 8 through 10 describe an alternative approach in which a convexification technique is used to directly convert energy minimization problems into linear programs without using graph cuts. The specific problem being solved using this method is the stereo matching problem. A major portion of the theory associated with solving the linear programs is common to both parts of the thesis.

A brief description of the contents of each chapter are as follows

- Chapter 2 describes the importance of graph cuts in the field of computer vision and shows how energy can minimized by repeatedly solving mincut problems.

- Chapter 3 discusses our formulation of the graph cuts problem as an unconstrained ℓ_1 norm minimization problem.
- Chapter 4 shows how to convert this ℓ_1 norm minimization into a linear programming problem and applies the barrier method to reduce the problem to that of solving a system of linear equations.
- Chapter 5 discusses methods of solving sparse systems of linear equations with focus on the conjugate gradient method which is our method of choice for this work.
- Chapter 6 introduces the ideas behind parallelizing the algorithm and then a specific implementation on the NVIDIA 8800 is shown.
- Chapter 7 presents results using the new graph cuts formulation on the problem of interactive foreground-background segmentation.
- Chapter 8 introduces the stereo matching problem, discusses relevant work in the area and formulates the stereo problem as a big linear program.
- Chapter 9 describes the solution to the stereo LP using interior point barrier method.
- Chapter 10 presents results using the new LP method for stereo on the standard Middlebury dataset.
- Chapter 11 concludes this thesis.

Chapter 2

Graph Cuts in Computer Vision

This chapter discusses the importance of graph cuts in computer vision. The chapter is organized as follows - Section 2.1 shows how energy minimization problems can be converted into mincut problems under certain constraints, Section 2.2 goes through some of the more traditional methods of finding a minimum cut in a graph and finally section 2.3 shows how these methods can be tuned in order to attain greater speeds on the kinds of graphs found in traditional vision problems.

2.1 Using Graph Cuts to solve Energy Minimization

This section describes how certain Energy Minimization problems can be reduced to the problem of finding a minimum cut on an associated graph. This was introduced in work by Boykov, Veksler and Zabih [12] and a more detailed discussion is provided in Kolmogorov and Zabih [58]. Let us first define the problem of finding a minimum cut on a graph.

We start with a directed graph $G = (V, E)$ in which each edge $(u, v) \in E$ has a positive weight $w(u, v)$ associated with it. If $(u, v) \notin E$ then $w(u, v) = 0$. Two vertices are distinguished in this graph. One is the source, denoted as s and the

other is the sink, denoted as τ [1]. Such graphs are called flow networks. A cut on the graph is defined as a partition of the vertex set into two subsets S and T such that

$$\begin{aligned} V &= S \cup T \\ S \cap T &= \phi \\ s &\in S \\ \tau &\in T \end{aligned} \tag{2.1}$$

Any edge (u, v) such that $u \in S$ and $v \in T$ is called a cut edge. The capacity of a cut, $C = \{S, T\}$ is defined as the sum of the weights of all the cut edges. This is denoted as $|C|$. Therefore the term minimum cut refers to a cut that has the least capacity amongst all possible cuts in the graph.

Graph cuts can readily be applied to binary labeling problems. That is, all pixels have the choice of only two labels, 0 or 1. An example of such a problem is foreground-background segmentation.

If each pixel can be labeled as only 0 or 1 the energy function in equation 1.1 can be rewritten as

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{ij}(x_i, x_j) \tag{2.2}$$

where the $x_i \in \{0, 1\}$ are the pixel labels.

From this energy function we would like to create a graph with a vertex set consisting of a node corresponding to every pixel, a source and a sink. This graph needs to satisfy the property that the value of the energy function for any assignment of labels to $x_1 \ldots x_n$ is equal to a constant plus the capacity of the minimum cut in the graph under the constraint that all nodes labeled 0 are placed into the S subset and all nodes labeled 1 are placed into the T subset. If one is able to create such a graph, the energy function is said to be graph-representable. If an energy function

[1] τ is being used to denote the sink since t will be used later for the barrier method.

is graph-representable it is clear that the minimum energy configuration is obtained by solving the min-cut problem on the associated graph and then labeling all the pixels in the S set as 0 and those in the T set as 1.

It turns out that whether or not an energy function of the form shown in equation 2.2 can be represented by a graph is dependent only on the pairwise smoothness functions E^{ij}. In their seminal work, Kolmogorov and Zabih [58] show that a certain property called regularity (which can be thought of as a generalization of triangle inequality) needs to be satisfied in order for an energy function to be graph-representable. The property is as follows

$$E^{ij}(0,0) + E^{ij}(1,1) \leq E^{ij}(0,1) + E^{ij}(1,0) \tag{2.3}$$

for every pair of neighboring vertices i and j. This criteria biases neighboring pixels towards taking on the same label.

The interesting aspect about the energy minimization formulations for most vision problems is that a large number of them do satisfy the regularity property. If regularity is satisfied the graph is constructed from the energy function by creating a graph for each term in the energy function and then merging them. This process is described in Algorithm 1. To see how it works, we create graphs as per the algorithm and see how they represent the values of the energy corresponding to different labelings.

Consider the figure 2.1, we see that in the graph on the left, if $v = 0$, this would mean the min-cut consists of $S = \{s, v\}$ and $T = \{t\}$. This would give it a cut capacity of 0. The value of the energy is $E(0)$. On the other hand if $v = 1$ the value of the energy is $E(1)$ and the value of the cut is $E(1) - E(0)$. Therefore in both cases the value of the energy function and the value of the cut capacity differ by the same constant $E(0)$ which implies that the data term is represented by the graph.

For the smoothness again referring to figure 2.1 there are 4 possible configurations for v_i, v_j. These are listed below with the corresponding cut capacities

Algorithm 1 Create graph corresponding to energy function defined in 2.2

Create a source s and a sink τ
for each term $E^i(x_i)$ **do**
  Add a vertex v_i corresponding to pixel i
  if $E^i(1) > E^i(0)$ **then**
    Create an edge from s to v_i of cost $E^i(1) - E^i(0)$
  else
    Create an edge from v_i to τ of cost $E^i(0) - E^i(1)$
  end if
end for
for each term $E^{ij}(x_i, x_j)$ **do**
  if $E^{ij}(1,0) - E^{ij}(0,0) > 0$ **then**
    Create an edge from s to v_i of cost $E(1,0) - E(0,0)$
  else
    Create an edge from v_i to τ of cost $E(0,0) - E(1,0)$
  end if
  if $E^{ij}(1,1) - E^{ij}(1,0) > 0$ **then**
    Create an edge from s to v_j of cost $E(1,1) - E(1,0)$
  else
    Create an edge from v_j to τ of cost $E(1,0) - E(1,1)$
  end if
  Connect v_i to v_j by an edge of cost $E(0,1) + E(1,0) - E(0,0) - E(1,1)$
end for

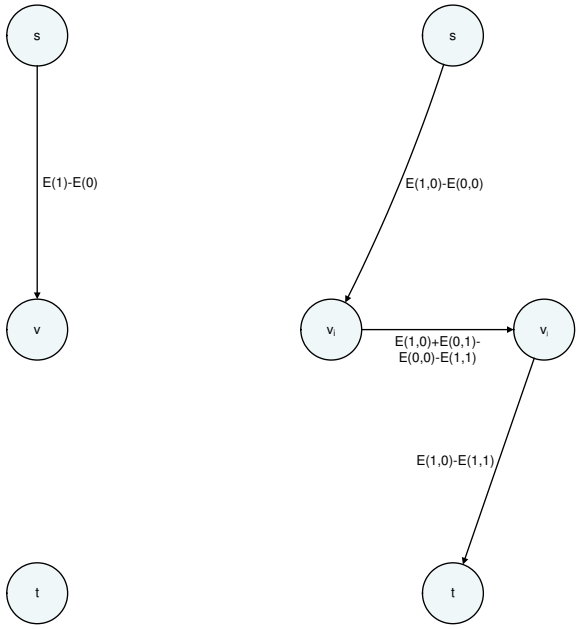


Figure 2.1: Conversion of energy functions to graphs such that solving the mincut problem provides a minimum energy configuration

$$\begin{aligned}
&\text{If} \quad v_i = 0, v_j = 0 \quad E(1,0) - E(1,1) \\
&\text{If} \quad v_i = 0, v_j = 1 \quad E(1,0) + E(0,1) - E(0,0) - E(1,1) \\
&\text{If} \quad v_i = 1, v_j = 0 \quad E(1,0) - E(1,1) + E(1,0) - E(0,0) \\
&\text{If} \quad v_i, v_j = 1 \quad E(1,0) - E(0,0)
\end{aligned} \tag{2.4}$$

It can be seen that in all these four cases, adding the constant $E(0,0) - E(1,0) + E(1,1)$ to the cut capacity will give us the value of the smoothness term. Thus following Algorithm 1 for energy functions that satisfy regularity will give us a graph such that the labeling corresponding to the minimum cut will minimize the energy.

From the algorithm, the graph created will have a source, a sink and a vertex for each pixel (or primitive). There are edges between neighboring vertices which for the remainder of this thesis will be referred to as *internal edges*. Also every vertex is connected to both the source and the sink. These edges will be referred to as the

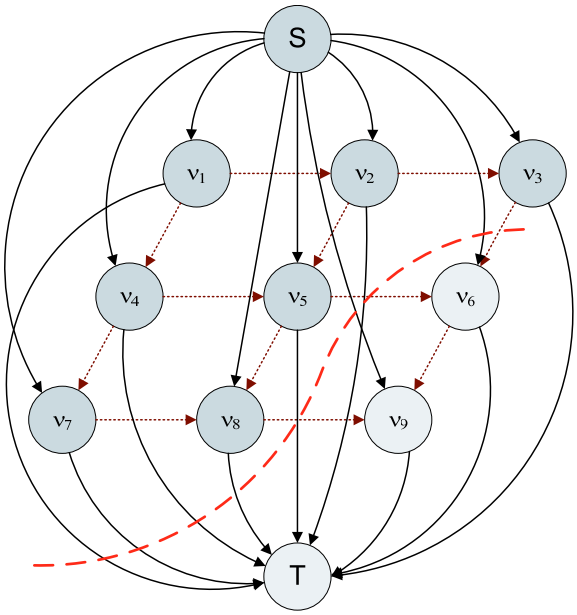


Figure 2.2: Typical grid graph in computer vision problems. The cut indicates which pixels should take on the value 1 and which should take on the value 0.

s-t edges. This therefore results in graphs that typically look like the one shown in figure 2.2.

2.1.1 Extension to multi-label problems

A multi label problem is reduced to a series of binary labeling problems by using the expansion move algorithm proposed in Boykov, Veksler and Zabih [12]. In general cases the problem is NP-hard. If the smoothness term is convex, then using Ishikawa [46]'s method an exact solution can be obtained via graph cuts.

When the smoothness term is metric, it is possible to compute a solution within a constant factor of the exact minimum. The method of computing this approximation is via the expansion move algorithm. Let us denote the energy associated with a label f by $E(f)$. Given any labeling f, a labeling f' is said to be an α expansion away from f if for all pixels, p such that $f_p \neq \alpha$, $f'_p = f_p$. Then expansion move

algorithm can then be described as in Algorithm 2.

Algorithm 2 Expansion move algorithm

1: Assume some current labeling f.
2: $Emin = E(f)$
3: **while** $\exists\alpha$ st $Enew = \min\{E(f') : f' \text{ is an } \alpha \text{ expansion away from } f\}$ satisfies $Enew < Emin$ **do**
4: $\quad Emin = Enew$
5: $\quad f = \arg\min\{E(f') : f' \text{ is an } \alpha \text{ expansion away from } f\}$
6: **end while**

The issue then becomes that of an efficient computational mechanism for finding the minimum energy labeling that is an α expansion away from the current labeling. We note that in an α expansion either the label of every pixel stays the same as before or it is changed to α. Hence the original energy function can now be viewed as a binary function taking an input of a binary vector corresponding to the primitives that are being labeled : a 0 means that the label stays the same and 1 means that the label was changed to α. So now if we look at equation 2.2 this basically means the following

$$\begin{aligned}
E^i(x_i) = D_i(f_i) \quad &\text{if} \quad x_i = 0 \\
E^i(x_i) = D_i(\alpha) \quad &\text{if} \quad x_i = 1 \\
E^{ij}(x_i, x_j) = V_{ij}(f_i, f_j) \quad &\text{if} \quad x_i = 0, x_j = 0 \\
E^{ij}(x_i, x_j) = V_{ij}(f_i, \alpha) \quad &\text{if} \quad x_i = 0, x_j = 1 \\
E^{ij}(x_i, x_j) = V_{ij}(\alpha, f_j) \quad &\text{if} \quad x_i = 1, x_j = 0 \\
E^{ij}(x_i, x_j) = V_{ij}(\alpha, \alpha) \quad &\text{if} \quad x_i = 1, x_j = 1
\end{aligned} \tag{2.5}$$

We can then use the method we discussed above for binary labeling and find the minimum energy labeling that is an α expansion away from the current. As noted in Algorithm 2, these expansion moves have to be done until no expansion move would lower energy. Therefore this involves solution of several graph cut problems.

Define a cycle to be a series of graph cut problems created by letting α vary over all the possible labels. Under the assumption that the data term and smoothness term are independent of image size, theoretically termination of the algorithm occurs in $O(P)$ cycles where P is the size of the image. However it has been empirically observed that convergence is attained in fewer cycles and that most of the changes occur during the first cycle.

2.2 Traditional methods for solving the mincut problem

In this section we briefly review two of the popular traditional algorithms for solving the min-cut problem. For a more detailed discussion on the subject the interested reader is referred to any algorithms text, for example [20].

Most approaches to the graph cut problem involve reducing it to what is known as the maximum flow problem. A flow in a weighted directed graph $G = (V, E)$ is a function $f : V \times V \rightarrow \mathbb{R}$ that satisfies

$$\begin{aligned} f(u,v) &\leq w(u,v) \\ f(u,v) &= -f(v,u) \\ \sum_{v \in V} f(u,v) &= 0 \end{aligned} \tag{2.6}$$

The value of a flow is defined as $|f| = \sum_{v \in V} f(s,v)$ which indicates the total flow out of the source. The problem of finding the maximum possible flow in a graph is called the maxflow problem.

The Maxflow mincut theorem relates the mincut problem to the maxflow problem. Say we have computed a maximum flow f then the minimum cut can be composed as follows. We define the residual capacity of an edge with respect to flow f as

$$w_f(u,v) = w(u,v) - f(u,v) \tag{2.7}$$

Then the residual graph G_f with respect to this flow is given by

$$G_f = (V, E_f)$$
$$E_f = \{(u,v) \in V \times V : w_f(u,v) > 0\} \tag{2.8}$$

Now in the graph G_f we find the vertices that can be reached from s, by performing a breadth first or depth first search. All these vertices constitute the set S of the cut. By definition, $T = V - S$.

The two standard algorithms for maxflow computation (and hence mincut computation) are the Ford-Fulkerson and Push-Relabel which are reviewed in the subsequent subsections.

2.2.1 Ford Fulkerson algorithm

The Ford Fulkerson class of algorithms [66] work in an iterative fashion with the goal in every stage being that of finding an augmenting path. An augmenting path is any path from the source to the sink along which it would be possible to push more flow. The algorithm is listed as Algorithm 3.

Algorithm 3 Maxflow using Ford-Fulkerson

```
1: choose an initial flow f of 0 on all edges
2: while there exists an augmenting path p from s to τ in residual graph G_f do
3:    c_f(p) = min{w_f(u,v) : (u,v) is in p}
4:    for all edges along the path p do
5:       f(u,v) = f(u,v) + c_f(p)
6:       f(v,u) = −f(u,v)
7:    end for
8: end while
```

where the graph G_f is created as per equations 2.7 and 2.8 . It can be shown

that the complexity of this algorithm is $O(|V||E|^2)$ [20] where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph.

In figure 2.3 an example of the Ford Fulkerson method being applied to compute maximum flow in a graph is provided

2.2.2 Push relabel algorithm

In the previous section we saw the Ford Fulkerson method which essentially looks at the whole graph during every iteration. Push relabel algorithms work on only one vertex at a time. Unlike Ford Fulkerson the constraint for internal vertices (not source or sink) that the incoming flow should be equal to outgoing flow is not always maintained. However the net flow at any vertex other than the source is non-negative.

There are two properties associated with each vertex. Firstly there is a positive excess flow at each vertex. Intuitively we think of this as an arbitrarily large reservoir attached to the vertex. Secondly there is a height associated with each vertex and this height increases as the algorithm progresses. We are only allowed to push flow from a higher vertex to a lower vertex. There are two basic operations that are performed which give the algorithm its name. The push operation takes excess flow from a vertex and pushes it to one of its neighbors. The relabel operation changes the height of a vertex.

Let $e(u)$ denote the excess flow at vertex u and $h(u)$ denote its height. Given an existing flow f, the push operation can be described as in Algorithm 4.

Algorithm 4 Push operation

1: **if** u is overflowing and has a neighbor v, $w_f(u,v) > 0$, $h(u) = h(v) + 1$ **then**
2: Push $d_f(u,v) = \min(e(u), w_f(u,v))$ from u to v.
3: Update $f(u,v)$, $e(u)$ and $e(v)$ and set $f(v,u) = -f(u,v)$.
4: **end if**

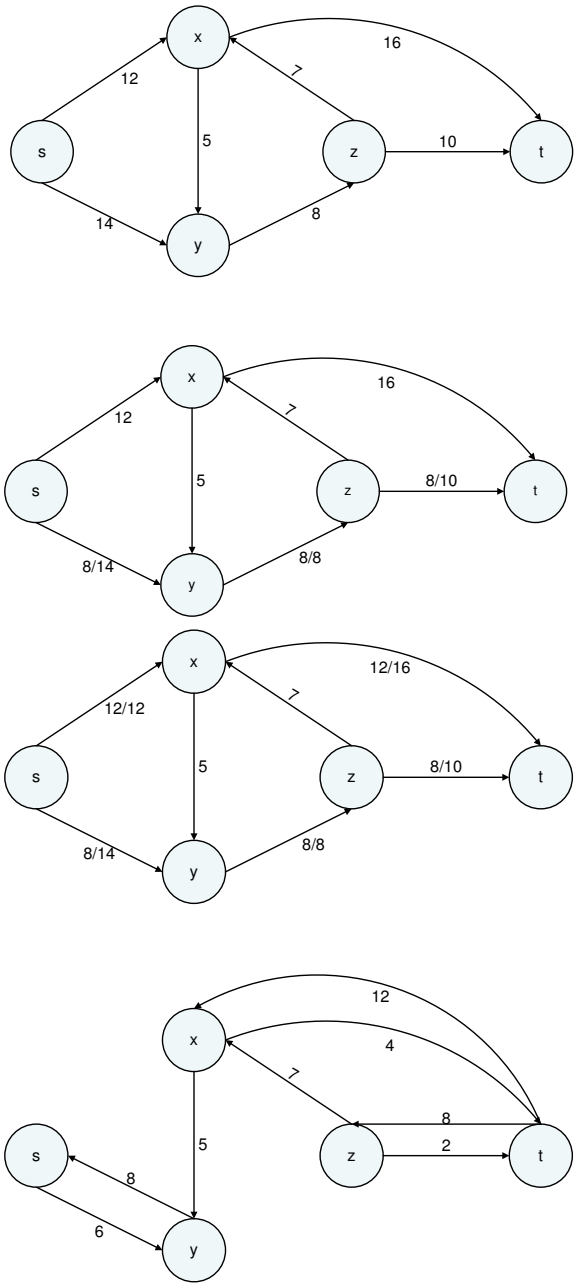


Figure 2.3: An example of the Ford Fulkerson method from [20]. The last figure shows the final residual graph and we see that there is no path from s to τ, and hence the algorithm terminates. s and y will be in the S subset and x, z and τ are in the T subset of the cut.

where the quantities $w_f(u, v)$ are computed according to equation 2.7. The relabel operation basically comes into play when we cannot find a neighbor,v, of an overflowing vertex,u, such that $h(u) = h(v) + 1$ and edge (u, v) can accommodate more flow. Then we need to increase the height of u by performing a relabeling as follows

$$h(u) = 1 + \min\{h(v) : (u, v) \in E_f\} \tag{2.9}$$

A generic push-relabel algorithm works by first performing an initialization as follows

Algorithm 5 Initialize push relabel

```
1: for all vertices u ∈ V do
2:     h(u) = 0
3:     e(u) = 0
4: end for
5: for all edges (u, v) ∈ E do
6:     f(u, v) = 0
7:     f(v, u) = 0
8: end for
9: h(s) = |V|
10: for all edges emerging from s do
11:     saturate flow along the edge
12:     update the excess flow at both the source and the other vertex on this edge
13: end for
```

The generic algorithm then works by doing a sequence of push and relabel operations as long as these operations are possible - as long as there is an overflowing vertex.

The generic push-relabel algorithm runs in $O(|V|^2|E|)$ time (again where —V— and —E— represent the cardinality of the vertex set and edge set respectively). It is possible to get $O(|V|^3)$ by doing a version of the push-relabel algorithm called the relabel-to-front algorithm which basically performs the push and relabel operations in a careful way. This algorithm maintains a list of overflowing vertices. The list is

scanned as the algorithm proceeds and the excess flow in each vertex in 'discharged'. The discharge operation chooses suitable push and relabel operations to ensure the entire excess flow is taken care of. If during the course of discharge, a vertex is relabeled, it is moved to the start of the list. The algorithm then treats the next vertex in the list and continues the procedure until every overflowing vertex has been discharged.

We illustrate this algorithm in figure 2.4 and 2.5 by applying it on the same graph shown for the Ford Fulkerson method(figure 2.3).

2.3 Adapting standard graph cut methods to suit vision problems

While the methods described in the previous section can be used in any kind of flow network, the class of problems that computer vision researchers are interested in almost always have structured graphs. In many cases, there is a good initial guess for the solution that could be suitably incorporated into the algorithm. This section looks at some of the ways in which traditional graph cut algorithms have been fine tuned in order to speed up the mincut computation in the realm of computer vision.

One of the most interesting aspects of graphs that show up in computer vision is that they all basically have the same structure - that of a grid. This fact has been exploited to provide substantial improvement over the two traditional algorithms we discussed in 2.2. Boykov and Kolmogorov [11] 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. The traditional Ford Fulkerson uses only one search tree from the source that is continuously rebuilt. Given $TREE_s$ from the source which consists of non-saturated edges from parent nodes to children and $TREE_t$ which has non-saturated edges from child to parent and includes the sink, the augmenting path is

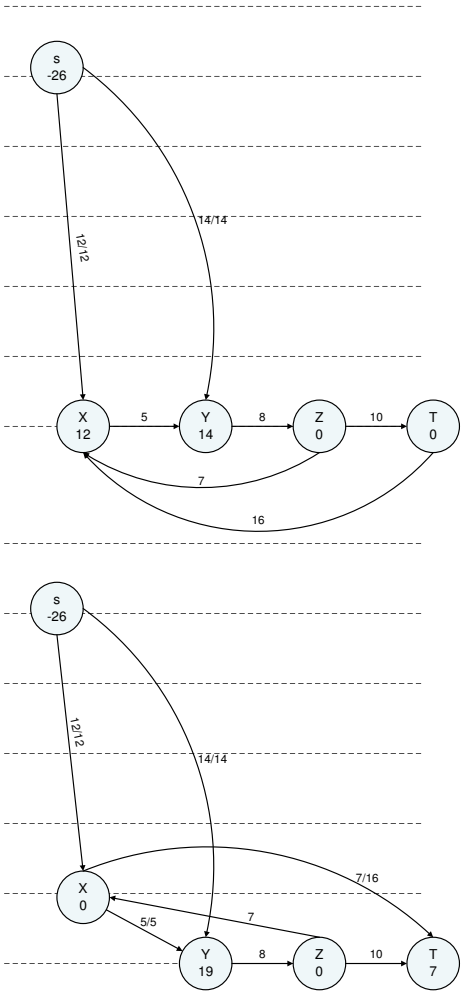


Figure 2.4: The example shown for Ford Fulkerson method being solved by the relabel-to-front algorithm. The numbers in each node indicate the current excess flow value. Each step discharges one of the vertices from a list. If at any point a relabel is needed during the course of discharge, the node is moved to the front of the list.

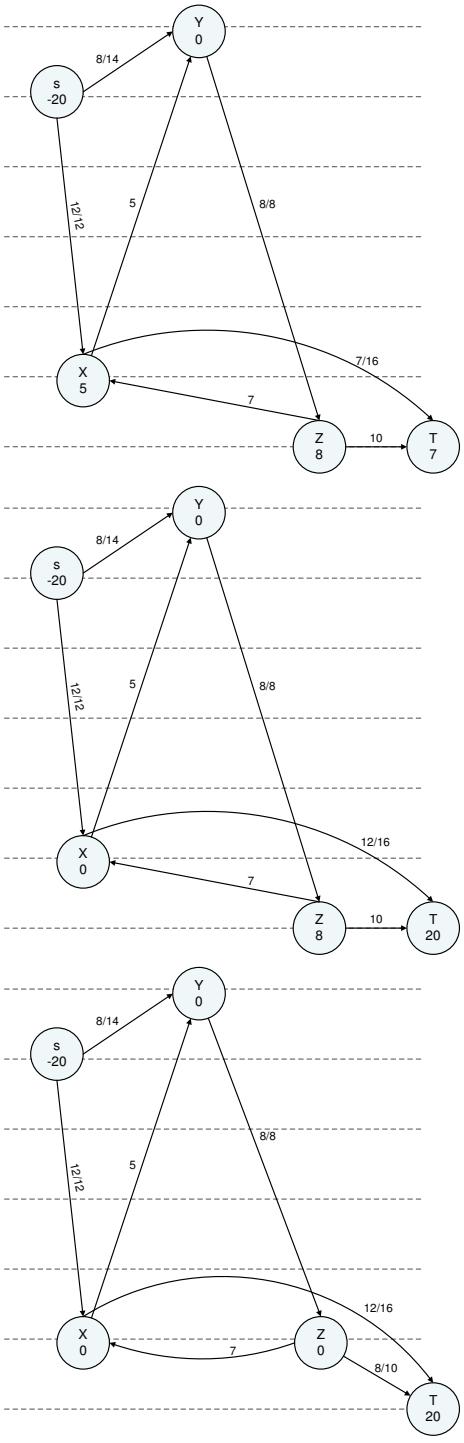


Figure 2.5: Final steps of the relabel-to-front algorithm shown in figure 2.4. At the final step the excess flow in each vertex becomes 0. At that point the excess flow value of the sink is the value of the maximum flow.

found as soon as the leaf node of one tree detects that it is connected to a leaf node of the other tree via an unsaturated edge. There are three stages in the algorithm which are repeated until maxflow is computed. In the 'growth' stage, the two trees are grown until they touch. In the 'augmentation' stage a bottleneck (minimum capacity) edge along the s-t path is found and that amount of flow is pushed through. The saturation of some of the edges along this path will create some nodes that belong to neither tree due to severance of the saturated edge. The final 'adoption' stage tries to restore the single-tree structure. Using this algorithm a mincut for the image restoration problem for a 512x512 image, for example, can be computed in 65 ms. This is about 10 times faster than Ford Fulkerson style algorithms and about 5 times faster than Push relabel style algorithms.

Kohli and Torr [55] use the idea of solution reusability while solving the object-background segmentation problem on video sequences. In video, the graph being created in order to perform the segmentation does not change too dramatically from frame to frame. So by taking the residual graph obtained in the terminating step of the previous step's mincut computation they change the weights to incorporate the new frame. This does lead to an illegal/infeasible flow sometimes and the paper describes how one can transform the graph to once again make the flow consistent. By performing updates as opposed to repeated creation of the graph, a speed up of 5-10 times was obtained over Boykov and Kolmogorov.

Juan and Boykov [49] also show how an initial solution can be used. However as opposed to updating flow, they update cuts. An initial non-feasible flow is created from the initial cut and this solution is then driven towards feasibility. The procedure is quite similar to Push-relabel. In push relabel one is only allowed to maintain excesses. Here both excesses and deficits are allowed at nodes. Given an initial cut C by saturating all the edges on the boundary between the source side and the sink side of the cut, they create excesses at the boundary nodes on the sink side of the cut and deficits at the boundary nodes on the source side of the cut. By using a two

tree approach similar to the one detailed in the preceeding paragraph the excesses get pushed towards the sink and the deficits are pulled by the source. This method, known as Active Cuts is suitable to be employed in a coarse to fine scheme where the cut at a coarse level is used as an initial solution for a finer level. By applying this to the problem of segmentation, they report a speed up of about 2 times. The scheme has also been applied to video segmentation where speed ups of 2-6 times were obtained.

Lombaert *et al* . [69] 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 approach is memory efficient as well as being faster. By using 2 levels in the hierarchy, the memory usage is 1/4th that used by non-hierarchical methods and the speed is 8 times faster.

Chapter 3

From Graph Cuts to ℓ_1 norm minimization

This chapter describes the process of reducing graph cuts to an unconstrained ℓ_1 norm minimization problem. This is done by manipulating the dual problem that arises from a linear programming model of the maxflow problem. The node-edge incidence matrix is introduced in Section 3.1 and then used to formulate the maxflow problem in a linear programming sense in Section 3.2. Section 3.3 shows how this new formulation still demands the same regularity conditions of the pairwise functions in order to be graph representable. The dual of the problem is introduced 3.4 and via some algebraic manipulations in 3.5 it is converted into an unconstrained ℓ_1 norm minimization.

3.1 The node edge incidence matrix

We begin by introducing our definition of the node-edge incidence matrix for a flow network $G = (V, E)$ with source s and sink t.

A small example of a graph with 4 nodes is shown in figure 3.1

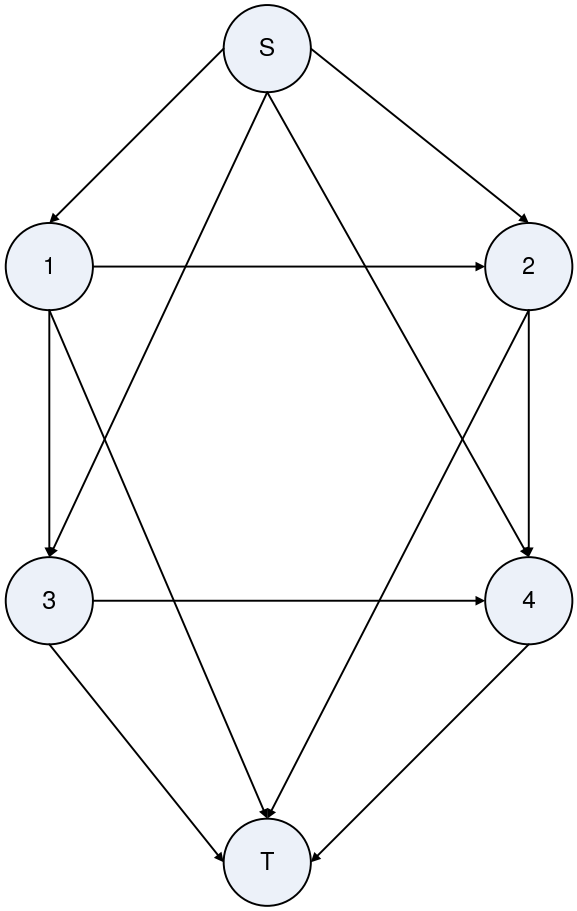


Figure 3.1: An example of a grid graph with 4 vertices whose incidence matrix A is given in Equation 3.1

.

The graph formulation we use is undirected. However, there is an arrow associated with each edge which is used to distinguish between positive and negative flows. Flow in the direction of the arrow is positive and flow in the opposite direction is negative.

The node edge incidence matrix which we denote by A, has size $n \times m$, where n is the number of internal vertices (vertices apart from the source and the sink) and m is the number of edges. 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 τ will only contain a single non-zero entry since there are no rows corresponding to the source and the sink. Hence for the above graph,

$$A = \begin{bmatrix} -1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{3.1}$$

This matrix A has interesting properties that are crucial in the development of the algorithm. Let us count the number of rows and columns of A. The number of rows will be equal to the number of pixels or primitives that we want to label. The important thing to remember is that there is no row for the source node or the sink node. Also if we refer to the grid graph in Figure 2.2 every pixel has at most 6 edges incident on it. Hence matrix A is a very sparse matrix with at most 6 non-zero entries per row. The sparseness will be of particular importance when we discuss the implementation of the algorithm. The number of columns will be at most n for the s-edges, n for the τ-edges and around $2n$ for the internal edges. Therefore the number of columns is approximately $4n$. For the image sizes that we deal with, 512×512, the matrix A has a quarter of a million rows and a million columns.

Without the sparseness of A, the algorithm would be completely infeasible. Also, because each interior node in the graph is connected to its four neighbours (and s and τ) it becomes very easy to compute the product of A or A^T with vectors; an operation that is required quite frequently in the algorithm.

With this definition of the node-edge adjacency matrix A in mind we turn our attention to formulating the max-flow problem as a linear programming problem.

3.2 Graph Cuts via linear programming

This section describes how the problem of graph cuts will be viewed in this thesis. Although there are some differences between this formulation when compared to traditional formulations, it results in the same regularity requirements which were shown in Chapter 2.

In the directed formulation of graph cuts, the maxflow mincut theorem [20] shows that an optimal solution to the maximum flow problem provides a solution to the minimum cut problem. The analysis performed in Cormen *et al* . [20]. extends to the undirected formulation being used. Therefore let us begin by formulating the maxflow problem as a linear program.

Like many combinatorial optimization problems, the maxflow problem can be reformated as a linear program. See for example Papadimitriou and Steiglitz [74]. One such formulation is shown in Equation 3.2.

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{c^T x} \\ \text{st} \quad & A\mathbf{x} = \mathbf{0} \\ & -\mathbf{w} \leq \mathbf{x} \leq \mathbf{w}. \end{aligned} \tag{3.2}$$

Here $\mathbf{x} \in \mathbb{R}^{\mathbf{m}}$ denotes a vector that indicates 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. Again, 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 use the node edge incidence matrix $A \in \mathbb{R}^{n \times m}$. 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 $\mathbf{x}$. Therefore, 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.

The main difference between this formulation and the Kolmogorov and Zabih formulation is that there is no direction associated with the edges in our formulation. In particular when the cut capacity is calculated *all* the edges that have nodes on opposite sides of the partition will contribute to the capacity. In the directed formulation, only those edges that go from the S side to the T side contribute in the cut capacity.

Despite this difference, the next section shows that one ends up with the same regularity requirement for the energy function to be graph representable.

3.3 Conditions for graph representability in our graph cut formulation

As pointed out before, the formulation in 3.2 differs slightly from the one presented by Kolmogorov and Zabih [58] which makes use of a directed graph. However, it can be shown that our formulation allows one to represent precisely the same set of energy functions involving up to pairwise clique potentials as the ones described in that work.

Recall from [58] that a function E of n binary variables, $x_1, \ldots, x_n$ is called graph representable if there exists a graph $G = (V, E)$ with terminals s and t and a subset of vertices $V_0 = \{\nu_1, \ldots, \nu_n\} \subset V - \{s, t\}$ such that for any configuration $x_1, \ldots, x_n$, the value of the energy $E(x_1, \ldots, x_n)$ is equal to a constant plus the cost of the minimum $s - t$ cut among all cuts $C = S, T$ in which $\nu_i \in S$, if $x_i = 1$, and $\nu_i \in T$ if $x_i = 0$.

We are dealing with energy functions of the following form

$$E(x_1, x_2 \ldots x_n) = \sum_{i=1}^{n} E(x_i) + \sum_{i=1}^{n} \sum_{j \in \mathrm{nbd}(i)} E(x_i, x_j) \tag{3.3}$$

Constructing the graph for the data terms is simple. For each of the pairwise terms, we would require a graph similar to the one in Figure 3.2. Let $w_1, \ldots, w_5$ represent non negative weights associated with the edges.

Let us consider what the mincut cost would be for various labelings of the interior nodes. Here the s node has a label of 1 and the τ node has a label of 0. Let the constant by which the cut cost and the energy value differ be given by K. Now consider what happens when both nodes are labeled 0. The value of the energy function is $E(0, 0)$. Since they are both labeled 0, they have to be on the T side of the cut. Under this constraint the capacity of the mincut becomes $w_4 + w_5$ which means for graph representability we need $E(0, 0) = w_1 + w_2 + K$.

In a similar manner for the other 3 configurations, we want the following to be

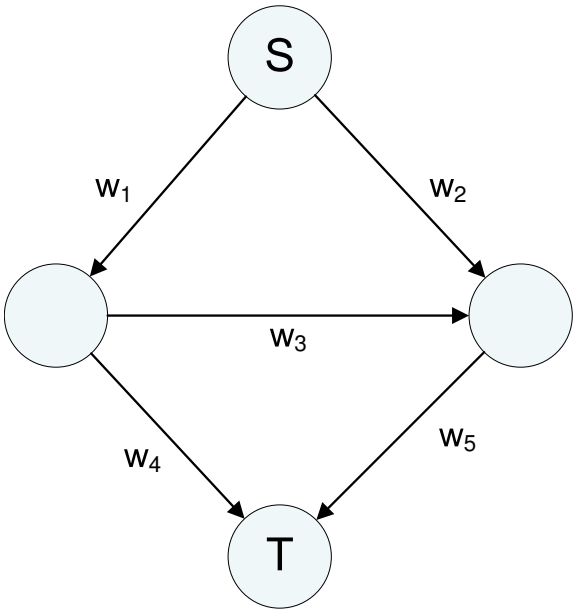


Figure 3.2: A two node graph used to determine the conditions for graph representability.

satisfied

$$\begin{aligned}
\begin{bmatrix} E(0,0) \\ E(1,1) \\ E(0,1) \\ E(1,0) \end{bmatrix} &= \begin{bmatrix} w_1 + w_2 + K \\ w_4 + w_5 + K \\ w_1 + w_3 + w_5 + K \\ w_2 + w_3 + w_4 + K \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ K \end{bmatrix}
\end{aligned} \tag{3.4}$$

By looking at these equations, since we are trying to express 4 numbers with seemingly 6 degrees of freedom we initially expect no restrictions on E. However, we note

that the following columns of the matrix - $(1,0,1,0),(1,0,0,1),(0,1,0,1),(0,1,1,0)$ and $(1,1,1,1)$ all lie in the perpendicular space of $(1,1,-1,-1)$. Hence any linear combination of these vectors will give rise to a vector that is perpendicular to $(1,1,-1,-1)$. Denote this vector as v. Then we have

$$\begin{bmatrix} E(0,0) \\ E(1,1) \\ E(0,1) \\ E(1,0) \end{bmatrix} = \lambda v + w_3 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{3.5}$$

Take the dot product of equation 3.5 with $(1,1,-1,-1)^T$

$$E(0,0) + E(1,1) - E(0,1) - E(1,0) = 0 - 2w_3 \tag{3.6}$$

now use the fact that w_3 is a positive edge weight to get

$$E(0,0) + E(1,1) - E(0,1) - E(1,0) \leq 0 \tag{3.7}$$

which is precisely the regularity condition. Therefore regularity is a necessary condition.

To see that it also is a sufficient condition, assume we have regularity. Then the following result is immediate

$$w_3 = \frac{E(0,0) + E(1,0) - E(0,0) - E(1,1)}{2} \tag{3.8}$$

Now we can assign the following values to the remaining weights, chosen to ensure equations 3.4 are satisfied and to ensure the positivity of all values apart from K.

$$
\begin{aligned}
w_2 &= \frac{E(1,0) + E(0,1) - E(1,1) + E(0,0)}{2} \\
w_5 &= E(0,1) \\
w_4 &= E(1,1) + E(1,0) \\
w_1 &= \frac{E(0,0) + E(1,0) + E(1,1) + E(0,1)}{2} \qquad (3.9) \\
K &= -E(1,0) - E(0,1) \qquad (3.10)
\end{aligned}
$$

Thus this gives a constructive proof that the undirected formulation handles the same energy functions as the Kolmogorov-Zabih formulations.

3.4 The dual problem

Instead of tackling the linear program described in Equation 3.2 directly, we proceed by formulating the associated dual problem. Slater's theorem [9] states that if the primal problem is convex and we can find a non-boundary point that is strictly feasible (the inequality constraints are satisfied as strict inequalities) then the dual problem's optimal value matches the primal problem's optimal value. Linear programs are always convex. Also it is easy to see that the solution $\mathbf{x} = 0$, the all zero vector, is a strictly feasible solution to 3.2. Therefore the conditions of Slater's theorem are satisfied and solving the dual problem will be equivalent to solving the primal problem.

Given a constrained optimization problem of the form

$$
\begin{aligned}
\min \quad & f_o(x) \\
\text{st} \quad & f_i(x) \leq 0 \quad i = 1, \ldots, m \\
& h_i(x) = 0 \quad i = 1, \ldots, p \qquad (3.11)
\end{aligned}
$$

the Lagrangian of the problem is obtained by associating a Lagrange multiplier with each constraint.

$$L(x, \lambda, \nu) = f_0(x) + \sum_i^m \lambda_i f_i(x) + \sum_i^p \nu_i h_i(x) \tag{3.12}$$

More specifically, the Lagrangian of the linear program described in Equation 3.2 has the following form

$$\begin{aligned} L(\mathbf{x}, \lambda, \nu) &= -\mathbf{c}^T\mathbf{x} + \nu^\mathbf{T}\mathbf{A}\mathbf{x} + \lambda_+^\mathbf{T}(\mathbf{x} - \mathbf{w}) \\ &\quad + \lambda_-^T(-\mathbf{x} - \mathbf{w}) \\ &= -\mathbf{w}^T(\lambda_+ + \lambda_-) \\ &\quad + \mathbf{x}^T(A^T\nu - \mathbf{c} + \lambda_+ - \lambda_-) \end{aligned} \tag{3.13}$$

In this expression the original primal objective function is augmented with multiples of the constraint functions ($\mathbf{x}$ - $\mathbf{w}$) ≤ 0, (-$\mathbf{x}$ - $\mathbf{w}$) ≤ 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:

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \tag{3.14}$$

Using this definition the Lagrangian dual of our problem is

$$g(\lambda, \nu) = -\mathbf{w}^T(\lambda_+ + \lambda_-) + \inf_x \mathbf{x}^T(A^T\nu - \mathbf{c} + \lambda_+ - \lambda_-) \tag{3.15}$$

since the first term does not involve $\mathbf{x}$, it is easy to see that

$$g(\lambda, \nu) = \begin{cases} -\mathbf{w}^T(\lambda_+ + \lambda_-) & \text{iff } A^T\nu - \mathbf{c} = (\lambda_- - \lambda_+) \\ -\infty & \text{otherwise.} \end{cases} \tag{3.16}$$

The dual problem is defined in terms of the Lagrangian dual function as

$$\begin{aligned} \max_{\lambda,\nu} \quad & g(\lambda,\nu) \\ \text{st} \quad & \lambda \geq 0 \end{aligned} \tag{3.17}$$

With this definition our original problem gives rise to the following dual problem

$$\begin{aligned} \min_{\lambda,\nu} \quad & \mathbf{w}^T(\lambda_+ + \lambda_-) \\ \text{st} \quad & A^T\nu - \mathbf{c} = (\lambda_- - \lambda_+) \\ & \lambda_+ \geq 0,\ \lambda_- \geq 0, \end{aligned} \tag{3.18}$$

where the maximization got turned into the minimization due to the negative sign.

3.5 From the dual problem to ℓ_1 norm

We now exploit the structure of the problem in Equation 3.18 to turn it into an ℓ_1 norm minimization. First note that the inner product is a sum of non-negative terms. $\mathbf{w}^T(\lambda_+ + \lambda_-) = \sum_{i=1}^{m} \mathbf{w}_i(\lambda_+ + \lambda_-)_i$. The weights w_i are fixed and are always non-negative. Hence minimization will involve minimizing each of the sums $(\lambda_+ + \lambda_-)_i$.

For a given value of ν, for each pair of $(\lambda_+)_i$ and $(\lambda_-)_i$ this minimization just involves solving the following basic problem

$$\begin{aligned} \min_{a,b} \quad & (a+b) \\ \text{st} \quad & a - b = c \\ & a \geq 0,\ b \geq 0 \end{aligned} \tag{3.19}$$

The solution of this problem is trivial but interesting. In the case when $c > 0$, the minimum value of $a + b$ will be c, corresponding to the solution $a = c$ and $b = 0$.

On the other hand when $c \leq 0$, $b = -c$ and $a = 0$ gives the minimum value of $-c$. Hence in both cases the minimum value is $|c|$.

Therefore it is clear that the minimum value that $(\lambda_- + \lambda_+)_i$ attains is $|(A^T\nu - \mathbf{c})_i|$.

This property allows us to reformulate the optimization problem in Equation 3.18 as follows:

$$\min_{\nu} \sum_{i=1}^{m} w_i |(A^T\nu - \mathbf{c})_i| \tag{3.20}$$

which can be rewritten as:

$$\min_{\nu} \|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1 \tag{3.21}$$

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:

$$\min_{\xi \in \{0,1\}^n} ||\text{diag}(\mathbf{w})(A^T\xi - \mathbf{c})||_1 \tag{3.22}$$

with the understanding that those nodes i such that $\xi_i = 0$ belong to the T set while those having $\xi_i = 1$ belong to the S set. This formulation is justified below.

Recall that for an edge to be a cut-edge, the two nodes that it is incident on have to be in different subsets, or in other words they have different labels.

Consider the structure of A^T. Since A is the node-edge adjacency matrix the transpose will have a row corresponding to every edge. There are 3 different kinds of edges and we consider them in turn

1. First we consider the rows that correspond to the internal edges. The corresponding component in the $\mathbf{c}$ vector will be zero. Each of these rows has one $+1$ entry and one -1 entry. Consider row i and let the $+1$ entry be in column

j and the -1 entry be in column k. Then while multiplying with ξ we have the following conditions. Either we have $\xi_j = \xi_k$ which basically implies the edge is not a cut edge in which case the resultant vector will have a 0 in the i^{th} component. Consequently the cost of this edge is not included in the value of the objective function. On the other hand if $\xi_j \neq \xi_k$ then we do have a cut edge and as we can see from the product the weight of this edge will be counted in the objective function and this is consistent with the edge weight contributing to the cut capacity.

2. Now if we consider those edges that emerge from the source, we know the corresponding component in $\mathbf{c}$ will be 1. The rows that correspond to these edges in A^T will have a single $+1$ in the row. Hence these edges will contribute to the objective function only if the node is labelled as 0 which would mean that the edge emanating from the source is a cut edge.

3. An argument analogous to the one used for the source edges shows that for the edges entering the sink, the only edges that contribute to the objective function value are those from a vertex j such that $\xi_j = 1$. By our convention, these vertices are therefore in the source set and the edges that emanate from them and enter the sink have to be counted towards cut capacity.

The significant difference between the objective function in Equation 3.22 and the one in Equation 3.21 is the absence of constraints on ν .

However, it is possible to show that the ν_i variables will converge to binary values without any external prodding.

Proposition 3.5.1. The unconstrained Lagrangian variables ν in the formulation given in 3.21 converge to either 0 or 1.

Proof. We first note that the unconstrained ℓ_1 norm minimization arose out of the dual problem derived from the linear programming formulation of the maxflow problem. In the optimal flow assignment for the grid graphs being dealt with, every vertex

is either connected to s or to t via a path consisting only of unsaturated edges. Let us call such a path an unsaturated path.

The Langrangians ν correspond to the labeling of the pixels. Let us label the source as 1 while the label corresponding to the sink is 0.

Now consider what happens when an edge i, connecting nodes j and k is unsaturated in the final optimal flow assignment. We apply the KKT conditions and in particular complementary slackness [9] to the formulation in the primal-dual pair of 3.2 and 3.18. The complementary slackness conditions basically say that when optimality is reached the product of the constraint and the Lagrange multiplier corresponding to that constraint will be zero. Therefore we get the following for edge i.

$$\begin{aligned} (x-w)_i \neq 0 &\Rightarrow (\lambda_+)_i = 0 \\ (-x-w)_i \neq 0 &\Rightarrow (\lambda_-)_i = 0 \\ &\Rightarrow (A^T\nu - c)_i = 0 \end{aligned} \tag{3.23}$$

This implies that if edge i is unsaturated, then the Lagrange multipliers associated with it will be 0. Now depending upon the type of edge that i is, we have three cases

1. $c_i = 1$ for an s edge. The corresponding row in A^T has a single $+1$ entry and therefore ν will have to be 1 for the pixel that is being connected to the source by this edge.

2. $c_i = 0$ for an internal edge. Here it can be seen that both pixels being connected by this edge will have to have the same label.

3. $c_i = 0$ for a t edge. The row in A^T has a single -1 entry and therefore the pixel that is being connected to the sink by this edge will have to be labelled 0.

Therefore all nodes connected to s by an unsaturated edge will be labeled 1 and this label propagates along every unsaturated edge. This leads to the conclusion that all nodes connected to s by an unsaturated path will be labelled 1 and all nodes connected to t by an unsaturated path will be labelled 0. When optimality is reached, every internal node has to belong to one of these two sets and hence every internal node is either going to be labelled 0 or 1. □

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. The unconstrained formulation in Equation 3.21 is advantageous in many ways. It underlines the connection between graph cuts and convex optimization and allows one to employ continuous optimization techniques that can exploit the structure of the problem. One can also use more liberal convergence criteria than are normally employed because all we require is for the $\nu_{\mathbf{i}}$ to be clearly above or below 0.5. Hence this results in fewer iterations for the interior point methods that are used to solve the problem.

Since ν_i have to be either 0 or 1, double precision calculations are not that important in the optimization. One can therefore take advantage of the abundance of single precision floating point operations available on architecture such as the graphics processing unit (GPU).

Chapter 4

Solving the ℓ_1 norm minimization

This chapter describes the implementation of a solution to the ℓ_1 norm minimization that is used in this thesis. The ℓ_1 norm minimization can again be reduced to a linear programming problem. This process is described in section 4.1. The solution to this formulation using interior point methods is discussed in 4.2.

4.1 Solving ℓ_1 norm minimization using linear programming

The unconstrained ℓ_1 norm minimization problem described in equation 3.21 can be formulated as a linear program by introducing an auxiliary variable $\mathbf{y} \in \mathbb{R}^m$ such that y_i serves as a placeholder for $|A^T\nu - \mathbf{c}|_i$. This can be done via the following optimization problem

$$\begin{aligned} \min \mathbf{y} \\ \mathbf{y} \geq (A^T\nu - \mathbf{c}) \\ \mathbf{y} \geq -(A^T\nu - \mathbf{c}) \end{aligned} \tag{4.1}$$

as described in [9]. It is easy to see why this works by considering two cases

of $A^T\nu - \mathbf{c}$ being positive, in which case the minimum value for y under the above constraints is $A^T\nu - \mathbf{c}$, while when $A^T\nu - \mathbf{c}$ is negative, then the minimum value will be $-(A^T\nu - \mathbf{c})$.

Again recalling that the weights w_i are all non-negative, we can incorporate $\mathbf{y}$ into the objective function of 3.21 by replacing $\|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1$ by $\mathbf{w}^T y$ and adding the constraints shown in 4.1. Koh, Kim and Boyd [54] use a very similar analysis to solve ℓ_1 regularized logistic regression problems.

The final linear program is shown below.

$$\begin{aligned} &\min \mathbf{w}^T\mathbf{y} \\ \text{st} \quad &\begin{bmatrix} A^T & -I_m \\ -A^T & -I_m \end{bmatrix} \begin{bmatrix} \nu \\ \mathbf{y} \end{bmatrix} \leq \begin{bmatrix} \mathbf{c} \\ -\mathbf{c} \end{bmatrix}. \end{aligned} \tag{4.2}$$

where I_m is the $m \times m$ identity matrix.

While the LP approach to ℓ_1 norm minimization overcomes the non-differentiability of the ℓ_1 it does introduce extra variables. An alternative method used by the optimization community is via the Huber M-estimator [68, 71, 75]. The Huber cost function is given by

$$\rho(t) = \begin{cases} \frac{1}{2}t^2, & \text{if} |t| \leq \gamma \\ \gamma|t| - \frac{1}{2}\gamma^2 & \text{otherwise.} \end{cases} \tag{4.3}$$

As γ is made smaller and smaller this approximates the ℓ_1 norm better and better. This cost function is a convex differentiable piecewise quadratic polynomial.

This approach was tried on the ℓ_1 norm problems that arose out of the graph cuts problem but the convergence rate was found to be extremely slow compared to the linear programming formulation.

4.2 Applying the barrier method to solve the LP

The problem formulated in equation 4.2 can be solved using the interior point method with logarithmic barrier potentials. In this approach a linear program which has inequality and equality constraints is approximated by a linear program with only equality constraints. The idea is to include the inequality constraints as logarithmic barrier potentials in the original objective function.

The logarithmic functions are being used as approximators for the indicator function

$$I(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0 \end{cases} \tag{4.4}$$

Using indicator functions the inequality constraints in the general optimization problem 3.11 can be pulled into the objective function as

$$\begin{aligned} \min \quad & f_o(x) + \sum_{i=1}^{m} I(f_i(x)) \\ \text{st} \quad & h_i(x) = 0 \qquad i = 1 \ldots p \end{aligned} \tag{4.5}$$

The problem with the indicator function is that it is not differentiable and hence one chooses to use the logarithm function $-(1/t)\log(-u)$ as an approximation for the indicator function $I(u)$. Figure 4.1 indicates this approximation for various values of t. The problem gets transformed into

$$\begin{aligned} \min \quad & f_o(x) + \sum_{i=1}^{m} -(1/t)\log(-f_i(x)) \\ \text{st} \quad & h_i(x) = 0 \qquad i = 1, \ldots, p \end{aligned} \tag{4.6}$$

Applying this to the problem, rewritten here

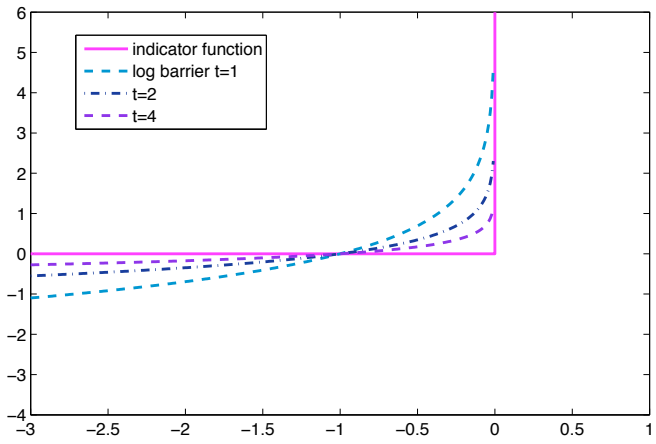


Figure 4.1: Approximations to the indicator function using different values of t

$$\begin{aligned} &\min \mathbf{w}^T\mathbf{y} \\ \text{st} \quad &\begin{bmatrix} A^T & -I \\ -A^T & -I \end{bmatrix} \begin{bmatrix} \nu \\ \mathbf{y} \end{bmatrix} \leq \begin{bmatrix} \mathbf{c} \\ -\mathbf{c} \end{bmatrix}, \end{aligned} \tag{4.7}$$

results in the following function that has to be minimized

$$\phi(\nu, \mathbf{y}) = t(\mathbf{w}^T\mathbf{y}) - \sum_{i=1}^{m} \log(\mathbf{y}_i - \mathbf{z_i}) - \sum_{i=1}^{m} \log(\mathbf{y}_i + \mathbf{z_i}) \tag{4.8}$$

where $\mathbf{z} = (A^T\nu - \mathbf{c})$ and we take t as a multiplicative factor for the objective function instead of dividing the barrier by t.

Therefore we now have a completely unconstrained convex optimization problem (since the negative of a log function is convex). By changing the value of t we get a family of such problems. The barrier method takes a sequence of increasing t values and solves the unconstrained minimization problems corresponding to them, using the last optimal point found as the starting point for the next unconstrained minimization problem.

An unconstrained minimization problem in which the objective function is convex and twice differentiable can be solved using Newton's method. The fundamental concept behind this iterative method is to take a step in each iteration in a direction that minimizes the second-order approximation of the function at the current point.

If a twice differentiable convex function $f(x)$ is to be minimized by this method, then in each iteration beginning with an estimate x , the step to be taken is in the direction

$$\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x) \tag{4.9}$$

By computing the Hessian and the gradient of our objective function we see that in each Newton iteration the system that we have to solve boils down to computing $[\Delta\nu\,, \Delta\mathbf{y}]^T$, in the following linear system:

$$\begin{bmatrix} AD_+A^T & AD_- \\ D_-A^T & D_+ \end{bmatrix} \begin{bmatrix} \Delta\nu \\ \Delta\mathbf{y} \end{bmatrix} = - \begin{bmatrix} A(\mathbf{d}_1 - \mathbf{d}_2) \\ t\mathbf{w} - (\mathbf{d}_1 + \mathbf{d}_2) \end{bmatrix} \tag{4.10}$$

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 the Schur complement [100] to factor out $\Delta\mathbf{y}$ the system can be further reduced to:

$$(A\text{diag}(\mathbf{d})A^T)\Delta\nu = -A\mathbf{g} \tag{4.11}$$

Where:

$$\mathbf{d}_i = 2/(\mathbf{y}_i^2 + \mathbf{z}_i^2) \tag{4.12}$$

and

$$\mathbf{g}_i = \frac{2\mathbf{z}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2} + \frac{2\mathbf{y}_i\mathbf{z}_i}{\mathbf{y}_i^2 + \mathbf{z}_i^2}(t\mathbf{w}_i - \frac{2\mathbf{y}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2}) \tag{4.13}$$

Once $\Delta\nu$ has been obtained from Equation 4.11, the $\Delta\mathbf{y}$ component of the step can

be computed using the following expression.

$$\Delta \mathbf{y} = D_+^{-1}((\mathbf{d}_1 + \mathbf{d}_2) - t\mathbf{w} - D_- A^T \Delta\nu) \tag{4.14}$$

The entire interior point optimization procedure is outlined in pseudo-code as Algorithm 6.

Algorithm 6 Calculate min-cut: $\min_{\nu} \|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1$

```
choose t,μ
ν_i = s_i/(s_i + τ_i)
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 4.12
      Compute g from Equation 4.13
      Solve (A diag(d) A^T)Δν = −Ag to get Δν
      Compute Δy from Equation 4.14
      If necessary, scale step by β so that ν + βΔν, y + βΔy are feasible.
   end while
   t = μ * t
end while
```

The input to this procedure is the vector of edge weights, $\mathbf{w}$. Given the way our graph is structured the node-edge adjacency matrix A and the indicator vector $\mathbf{c}$ stay the same for each problem.

The parameters to be chosen in this algorithm are t, μ and the two thresholds for convergence. In current empirical analysis, setting the thresholds to be relative changes of 1 percent works for most of the examples. As detailed in Boyd and Vandenberghe, the choice of μ is not that critical and we have experimented with values ranging from 2 to 15 and have obtained similar results for the different values of μ. The choice of t is more interesting. We first compute a feasible solution for the flow formulation by taking the minimum of the s links and τ links incident on each node. Then given an initial dual feasible solution (any labelling will be feasible), we compute the duality gap η which is the difference between the dual objective

function value and the primal objective function value. Then the initial t can be chosen as m/η as suggested by Boyd and Vandenberghe. While this initial choice of t does work, one can obtain better convergence by hand-selecting t values. A method of automatically generating a t value, given an initial solution, is still under investigation.

As an initial solution to our interior point method we use $\nu_i = s_i/(s_i+\tau_i)$, where s_i and τ_i are the weights of the s-edge and τ-edge incident on node i respectively. This is indicative of a purely data driven labelling of the pixels and provides a starting point in which all labels are between 0 and 1. While this is a good starting point, a completely uninformative initial labeling which assigns 0.5 to every pixel, takes only around 20 extra conjugate gradients (the iterative method of choice for solving the normal equations in this thesis).

Note that the principal step in this procedure is the solution of the sparse linear system given in Equation 4.11 and step 8 of the algorithm. These linear equations are exactly the same as the ones that arise out of the solution to the following weighted least squares problem

$$\min \|\mathrm{diag}(d)^{1/2}(A^T\Delta\nu + \mathrm{diag}(d)^{-1}g)\|_2 \tag{4.15}$$

The system of equations used to solve least squares problems are referred to as **normal** equations and we are going to borrow that terminology to refer to 4.11. This means that the original ℓ_1 norm minimization problem has been reduced to the problem of solving a sequence of sparse linear systems. The next chapter deals with methods of solving these normal equations.

Chapter 5

Solution of the normal equations

This chapter details the method used to solve the normal equations - the system of linear equations in 4.11.

We begin by discussing important properties of the matrix $A\text{diag}(\mathbf{d})A^T$ in section 5.1. This matrix will be denoted as L from this point on. Section 5.2 then presents the various methods that can be used to solve the normal equations, with particular emphasis on the conjugate gradients method which is our method of choice for this thesis. All the methods are discussed in greater depth in Golub and Van Loan's classic text [33].

5.1 Properties of $A\,\mathbf{diag}(\mathbf{d})\,A^T$

Throughout the thesis we will use a lot of properties of $\mathcal{L} = AA^T$. This matrix is an $n \times n$ matrix that represents node-node adjacency in the following manner. The diagonal entries give the degree of the corresponding vertex. The off diagonal entries are either 0 or -1. $\mathcal{L}(i,j) = -1$ indicates that nodes i and j are connected. This matrix, like the matrix A is extremely sparse. Since each vertex has at most 4 neighbours (6 in a 3d case) we have at most 5 non-zero entries per row.

For the example in 3.1

$$\mathcal{L} = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \tag{5.1}$$

At this point we note that the matrix $\mathcal{L} = (AA^T)$ corresponds to the graph Laplacian where the rows and columns associated with the s and t nodes removed. L is nothing but a weighted version of this matrix. Matrix L is symmetric by construction and, since the entries in $\mathbf{d}$ are all positive, it is also positive definite.

The entries in $\mathbf{d}$ can be viewed as weights of the edges of the graph that are dependent on the current node labeling. With each change in the labels assigned to the nodes, these weights change.

The entries along the diagonal of the matrix L_{ii} correspond to the sum of the current weights of the edges impinging on the node i 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 current weight of the edge connecting nodes i and j.

Definition 5.1.1. We say that $A \in \mathbb{R}^{n\times n}$ is strictly diagonally dominant if

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad i = 1 \ldots n$$

Now, due to the inclusion of the current weights of the s links and t links (both of which have non negative weights) in the diagonal entries, it is clear that the matrix L will be strictly diagonally dominant. So far we have shown that L is sparse, symmetric, positive definite and strictly diagonally dominant. It also happens to be highly structured.

$$L = \begin{bmatrix} D_1 & F_1 & & \cdots & 0 \\ F_1 & D_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & F_{n_1-1} \\ 0 & \cdots & & F_{n_1-1} & D_{n_1} \end{bmatrix} \tag{5.2}$$

that is, it is block tridiagonal, where each of the D_i blocks is tridiagonal and each of the F_i blocks is diagonal. Recalling that the graph we work on is a grid, let there be n_1 rows and n_2 columns, $n = n_1 n_2$. Then L is $n \times n$. Each block D and F is $n_2 \times n_2$ and there are n_1 blocks of D and $n_1 - 1$ blocks of F.

Such matrices come up very frequently while solving partial differential equation problems on a rectangular grid. See for example Farlow's text on partial differential equations [25].

A classic example is the Poisson differential equation that is encountered extensively in areas like mechanics and electrostatics

$$\nabla^2 \phi = f \tag{5.3}$$

If we are solving this equation on a rectangular domain, we get

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f \tag{5.4}$$

If we discretize the domain by creating m nodes along the height of the rectangle and n along the width and then proceed to approximate the partial derivatives at each node by using the vertical and horizontal neighbours of that node, [1] we get the system of equations

$$A\phi = f \tag{5.5}$$

[1] The approximation of the partial derivative could be done by using 8 neighbours instead of 4, resulting in a structurally different matrix

where

$$A = \begin{bmatrix} D & F & & \cdots & 0 \\ F & D & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & F \\ 0 & \cdots & & F & D \end{bmatrix} \tag{5.6}$$

the D matrices are tridiagonal of size n by n and $F = -I_n$, the negative of the identity matrix. Each D matrix looks like the one shown in 5.7 and there are m of them and $2(m-1)$ off diagonal blocks.

$$D = \begin{bmatrix} 4 & -1 & & \cdots & 0 \\ -1 & 4 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \cdots & & -1 & 4 \end{bmatrix} \tag{5.7}$$

The matrix A has a sparseness structure that is exactly the same as our matrix L. Therefore there is an interesting connection between our problem and the solution of such PDEs. The main difference between L and A is that all the diagonal blocks are equal and all the off-diagonal blocks are equal in the Poisson system. This is not going to be the case in our system.

To summarise, the matrix that we are dealing with, L, is sparse, banded tridiagonal, symmetric and positive definite. All these properties result in it being an ideal candidate for a number of linear equation solvers and we explore them in the next section.

5.2 Solving a large sparse system of linear equations

The set of methods used to solve a sparse system can be broadly classified into two categories.

1. **Direct methods** - These are methods in which the matrix is factorized into a form that makes it simpler to solve the equations. Cholesky decomposition and cyclic reduction are examples.

2. **Iterative methods** - In these methods, an initial guess is made for the solution and through a process of iterative refinement steps, the guess is made to converge to the actual solution. Gauss-Seidel, Jacobi, SOR(successive over relaxation) and Conjugate gradients are examples of this.

5.2.1 Cholesky decomposition

One of the popular methods of solving a system of linear equations that is symmetric and positive definite is to use the Cholesky decomposition. If $A \in \mathbb{R}^{n \times n}$ then it is always possible to find a unique lower triangular matrix $G \in \mathbb{R}^{n \times n}$ such that $A = GG^T$. Solving a system of equations involving an upper triangular or a lower triangular matrix just involves back or forward substitution respectively and therefore the factorization step leads to a much simpler set of equations.

The factors of a block tridiagonal matrix will be block bi-diagonal, however there is no guarantee of sparseness within the blocks. Our original matrix has n_1 tridiagonal blocks of size $n_2 \times n_2$ on the diagonal. The Cholesky decomposition could therefore result in a storage requirement of $n_1 n_2^2$ entries which is extremely memory intensive.

Also the factorization is not amenable to parallelization since it involves forward and back substitution in which the current step depends on the result from the

preceeding step.

5.2.2 Cyclic Reduction

Cyclic reduction is a typically used for tridiagonal or block tridiagonal systems. This method is discussed in Gander and Golub [30]. The main idea is to eliminate either the even or the odd unknowns(blocks) by expressing them in terms of the other half. The resultant equations are regrouped and again half the variables are eliminated. Finally we end up with only one block or one variable that needs to be solved for. Once this is done the intermediate equations get solved in a bottom up manner.

This is best illustrated with an example. We reproduce the one given in Golub and Van Loan but show only the first step. The subsequent reduction steps are done in exactly the same manner.

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \\ 12 \\ 14 \end{bmatrix} \tag{5.8}$$

By taking sets of 3 equations and eliminating the odd variables we get

$$\begin{aligned} -14x_2 + x_4 &= -24 \\ x_2 - 14x_4 + x_6 &= -48 \\ x_4 - 14x_6 &= -72 \end{aligned} \tag{5.9}$$

In the next step we can eliminate x_2 and x_6 and get a single equation in terms of x_4 and then work backwards to get the other variables.

As can be seen above, this algorithm works well for tridiagonal systems. This algorithm has been implemented in parallel by Kass and Lefohn and Owens [50] on the GPU for interactive depth of field effects in computer graphics. Our system is block-tridiagonal and applying the method directly is a little cumbersome. However, as we will see in Section 5.2.4 this method can be used to solve the tridiagonal systems that arise during the course of preconditioning the matrix.

5.2.3 Jacobi and Gauss-Seidel Iterative methods

The Jacobi method and the Gauss-Seidel method are both specific members of a general class of iterative algorithms for solving a linear system. The idea is to break the matrix A up into $A = M - N$ and iteratively solve the system of equations

$$Mx^{(k+1)} = Nx^{(k)} + b \tag{5.10}$$

where the matrix M is chosen such that it is easy to solve a linear system with M as the matrix.

In the Jacobi method M is a diagonal matrix whose diagonal entries match those of A.

In the Gauss-Seidel method M is obtained by extracting the lower triangular portion of A.

Theorem 5.2.1. *If $A \in \mathbb{R}^{n \times n}$ is symmetric and positive definite then the Gauss-Seidel iteration converges for any initial solution.*

Proof. A proof for this theorem can be found in Golub and Van Loan [33]. □

The above theorem ensures the applicability of this method to our system of equations, These methods are easily parallelizable but the rate of convergence depends on the spectral radius of $M^{-1}N$ and in experiments they required many iterations to converge.

5.2.4 Conjugate gradients

The numerical properties of the matrix L make the resulting linear system amenable to solution by the method of conjugate gradients [33]. An excellent introduction on the subject of conjugate gradients can be found in Shewchuk's article [82].

Assume the system of equations we are trying to solve is $Ax = b$, where A is symmetric positive definite of size $n \times n$. The solution to this can be found by minimizing the function $\phi(x) = x^T Ax - x^T b$. One method of doing this would be to perform steepest descent by successively choosing the negative of the gradient as the search direction. However the rate of convergence of this method becomes very slow if the ratio λ_1/λ_n, of the largest and smallest eigenvalues becomes very large. As an alternative, in the conjugate gradient method the search directions chosen are what are known as A-conjugate directions. The set of search directions $\{p_1, p_2, \ldots p_k\}$ satisfies $p_i^T A p_j = 0$ for all $i \neq j$. Given a set of previous search directions $\{p_1, p_2, \ldots, p_{k-1}\}$, the direction p_k is chosen such that it is A-conjugate to all these directions and is as close as possible to the negative of the gradient at the current point. By following this procedure, convergence is guaranteed in n steps.

Pseudocode for the conjugate gradients method is presented in Algorithm 2.

Algorithm 7 Solve $A\mathbf{x} = \mathbf{b}$ using Conjugate Gradients

$\mathbf{x} = \mathbf{x_0}$
$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$
$\mathbf{p} = \mathbf{r}$
$\rho = \mathbf{r^T r}$
while $\sqrt{\rho}/||\mathbf{b}|| > \epsilon$ **do**
    $\mathbf{a} = \mathbf{A}\mathbf{p}$
    $\alpha = \rho/(\mathbf{a^T p})$
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
    $\mathbf{r} = \mathbf{r} - \alpha\mathbf{a}$
    $\rho_{\text{new}} = \mathbf{r^T r}$
    $\mathbf{p} = \mathbf{r} + (\rho_{\text{new}}/\rho)\mathbf{p}$
    $\rho = \rho_{\text{new}}$
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 scalar multiplication and addition (also called SAXPY) operations. All of these operations are amenable to implementation on the parallel architectures found on modern GPUs and multi-core processors [8, 61]. We discuss the GPU and our implementation of the algorithm on an NVIDIA 8800 in the next chapter.

While the conjugate gradient method converges for any symmetric, positive-definite matrix, the rate of convergence is dependent on the condition number of the matrix, κ. To reduce the norm of the error by a factor of ϵ we need

$$i \leq \lceil \frac{1}{2}\sqrt{\kappa}\ln(\frac{2}{\epsilon}) \rceil \tag{5.11}$$

iterations.

An important theorem concerning the convergence rate of the conjugate gradient method is as follows

Theorem 5.2.2. *If $A = I + B$ is an n-by-n symmetric positive definite matrix and rank(B)=r then the conjugate gradient method converges in at most $r + 1$ iterations.*

The conjugate gradient algorithm can therefore 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 Golub and Van Loan [33]. 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 [19] describe preconditioning strategies that are specifically designed for the types of matrices that we seek to invert. The idea is to perform an incomplete block Cholesky factorization as opposed to an exact Cholesky factorization on the block tridiagonal matrix. The Cholesky factors of such matrices are block bidiagonal but the individual blocks are no longer sparse and hence an exact Cholesky becomes an intractable task.

As an example we see what happens when we try and compute the exact Cholesky decomposition of the following block tridiagonal matrix where the A_i are tridiagonal and E_i are diagonal that is structurally similar to ADA^T

$$A = \begin{bmatrix} A_1 & E_1^T & 0 \\ E_1 & A_2 & E_2^T \\ 0 & E_2 & A_3 \end{bmatrix} \tag{5.12}$$

Let the exact factorization be given by

$$G = \begin{bmatrix} G_1 & 0 & 0 \\ F_1 & G_2 & 0 \\ 0 & F_2 & G_3 \end{bmatrix} \tag{5.13}$$

If we were to solve for the blocks of G we would get the following system of equations

$$\begin{aligned} G_1 G_1^T &= B_1 \equiv A_1 \\ F_1 &= E_1 G_1^{-T} \\ G_2 G_2^T = B_2 \equiv A_2 - F_1 F_1^T &= A_2 - E_1 B_1^{-1} E_1^T \\ F_2 &= E_2 G_2^{-T} \\ G_3 G_3^T = B_3 \equiv A_3 - F_2 F_2^T &= A_3 - E_2 B_2^{-1} E_2^T \end{aligned} \tag{5.14}$$

The problem now is that although B_1 is tridiagonal and hence easy to factorize, B_1^{-1} is not tridiagonal and hence B_2 will not be tridiagonal and therefore much harder to factorize. So the incomplete preconditioning strategy is to approximate each of the B_i^{-1} by a tridiagonal matrix. Concus,Golub and Meurant discuss three methods to do this. The simplest of these is to just approximate B_i^{-1} by a diagonal matrix whose elements are $\frac{1}{(B_i)_{jj}}$ and we use this approximation in our implementation of their preconditioning strategy.

Chapter 7 presents results that illustrate how effective these strategies can be in improving the convergence rate of the solver. The reader will see how in the case of foreground background segmentation usage of the Concus and Golub preconditioner can reduce the number of iterations by a significant factor when compared to the unconditioned system. The preconditioner involves the solution of a series of matrix equations involving tridiagonal matrices. To speed up the process of solving these equations, the cyclic reduction method seen in Section **??** can be used.

Experiments were also carried out using a simpler preconditioning strategy where the matrix C is chosen as follows $C = \text{diag}(\mathbf{a})$, $\mathbf{a}_i = 1/\sqrt{A_{ii}}$. This is known as the Jacobi 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 this preconditioner does not affect the fill pattern of the matrix.

Chapter 6

Implementation strategies

In the previous chapters we have described the process of solving the graph cut problem by converting it into an unconstrained ℓ_1 norm minimization problem which in turn gets converted into a linear programming problem. The main advantage of this conversion is that the solution of the linear program consists of operations that are readily parallelizable. This chapter shows how each of the basic functions used in the algorithm can be parallelized.

We also discuss one particular parallel implementation of the algorithm, on a GPU - the NVIDIA GeForce 8800.

6.1 Parallelizing the ℓ_1 minimization

If we look at Algorithms 6 and 7, the basic operations that we need are Scalar multiplication and addition (SAXPY), pointwise multiplication, computing the ℓ_1 norm of a vector, inner products, computing the minimum of a vector and finally the matrix vector products $A\text{diag}(d)A^T\Delta\nu_i$ (corresponding to the i^{th} iteration of conjugate gradients) , Ag and $A^T\nu$. The bottleneck step, as mentioned before, is the conjugate gradient solution to the sparse system of equations. Conjugate gradients have been the subject of a lot of research in the parallel computing community [3,

18, 42].

The pointwise operations such as SAXPY and pointwise multiplications are trivially parallelizable. Computing the ℓ_1 norm, the sum of all elements of a vector and the minimum element of a vector are all called reduction operations as the final outcome is a vector being reduced to a single scalar. To parallelize these operations, the idea is to get processors/threads to perform the operation on mutually exclusive sections of the vector and keep combining the partial results. Algorithm 8 below shows how to perform the summation of all elements of a vector in a multi-threaded approach.

Algorithm 8 Adding n elements using n threads

```
1: Zero initialize an array partialsums of size n.
2: sliceindex=n/2
3: while sliceindex ≥ 1 do
4:    if threadindex ≤ sliceindex then
5:       partialsums[threadindex]=partialsums[threadindex+sliceindex]
6:    end if
7:    sliceindex=sliceindex/2
8:    synchronize threads
9: end while
```

So in the end **partialsums**[0] will have the resultant sum.

The matrix vector products all involve highly structured sparse matrices and hence can be computed by just a few multiplication and addition operations per element, as opposed to dense matrix multiplication.

For example $L = A\,\mathrm{diag}(\mathbf{d})\,A^T$ as discussed before has at most 5 non-zero entries per row. 1 corresponding to the diagonal entry and 4 off-diagonal entries corresponding to the 4 neighbours of the node. Referring to figure 6.1 the matrix vector multiplication boils down to the following computation being done on each node

$$(A\ \mathrm{diag}(d)\,A^T\nu)_i = (w_s{+}w_t{+}w_N{+}w_S{+}w_E{+}w_W)\nu_i{-}(w_N\nu_N{+}w_S\nu_S{+}w_E\nu_E{+}w_W\nu_W) \tag{6.1}$$

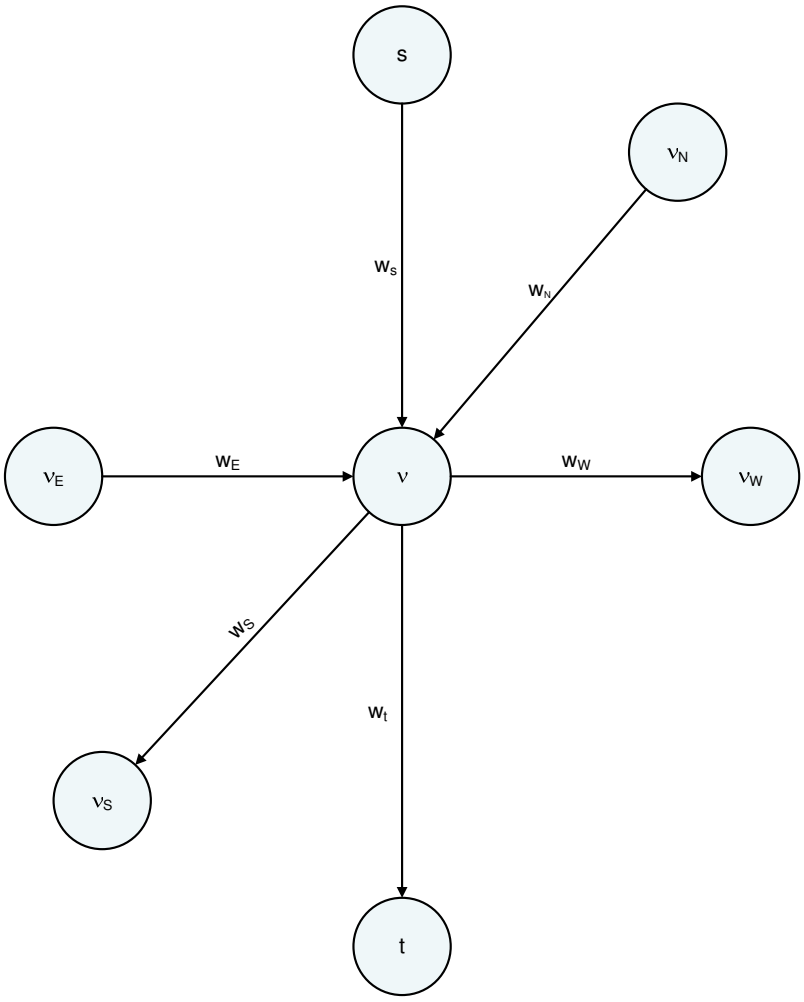


Figure 6.1: Computation of $ADA^T\nu$ for the grid graphs using only 6 operations per element

where the ws are the current weights of the edges (as described earlier in Section 5.1 the entries in $\mathbf{d}$ can be viewed as weights of the edges of the graph that are dependent on the current node labelling.). As can be seen from the equation, it basically involves communication between the current node and its four neighbours and hence can be parallelized easily.

In the next section we discuss our implementation on one of the possible parallel architectures - the NVIDIA 8800. As noted before, the algorithm is parallelizable and is implementable on a host of other architectures as well and it will be interesting to see how the evolution of hardware affects the performance of this algorithm.

6.2 Implementation on the NVIDIA 8800

In the previous section we gave a generic description of how our algorithm can be parallelized. This subsection presents one such parallel implementation, on a GeForce 8800 GPU from NVIDIA.

With the increasing programmability of graphics processing units (GPUs), researchers were able to use them for more than the computations in graphics for which they were designed. In the past, this was done using shader languages like Cg [27] which is NVIDIA's shader language, GLSL [77] which works alongside OpenGL, and HLSL [38] which works with DirectX on Windows systems. With the GeForce 8800 though, NVIDIA introduces a new programming framework called CUDA (Compute Unified Device Architecture), which makes general purpose programming a lot easier by treating the GPU as a coprocessor and off-loading computations to the GPU in a manner similar to multi-threaded function calls.

To understand how the GPU works, it is insightful to look at the modern day hardware organization of the GPU as shown in 6.2. There are a number of multiprocessors, each of which contains some amount of processors. Each processor has its own set of registers, in a manner very similar to a CPU. However the shared memory

which acts a lot like the CPU data cache is shared across all processors within one multiprocessor and data can be accessed in parallel. Accessing data from the shared memory is a lot faster than the device memory. Device memory is basically the DRAM corresponding to the GPU and is accessible to all the multiprocessors.

When the GPU is being used for general purpose computations, it is essentially viewed as a co-processor to the CPU. Those instructions that fit into the SIMD (Single Instruction Multiple Data) model get off-loaded onto the GPU. Basically, a portion of the application that is run many times but independently on different data is compiled to the instruction set of the GPU and sent to it for execution. The task to be performed on the GPU is divided up among the multiprocessors and at any given clock cycle, each processor of the multiprocessor is performing the same instruction, but on different data. Effective programming requires maximum utilization of the shared memory as opposed to the device memory.

We now discuss some of the applications that have been ported successfully onto the GPU.

6.2.1 A survey of General Purpose Computing on the GPU

A whole community of research has evolved around the field of general purpose computing on the GPU and an excellent survey paper on the topic can be found in Owens *et al* [73]. The website http://www.gpgpu.org is a great resource for the latest happenings in the area.

As noted earlier, applications that follow the SIMD paradigm can be performed very fast on the GPU. As a trivial example, adding two matrices simply involves performing the same *add* operation on multiple data pairs consisting of an element of the first matrix and the corresponding element from the second matrix. Not all operations are that easily translated into the SIMD paradigm though and the skill in GPGPU lies mainly in converting traditional/CPU based algorithms into a possibly new algorithm that is more favourably executed on the GPU. An example of this is

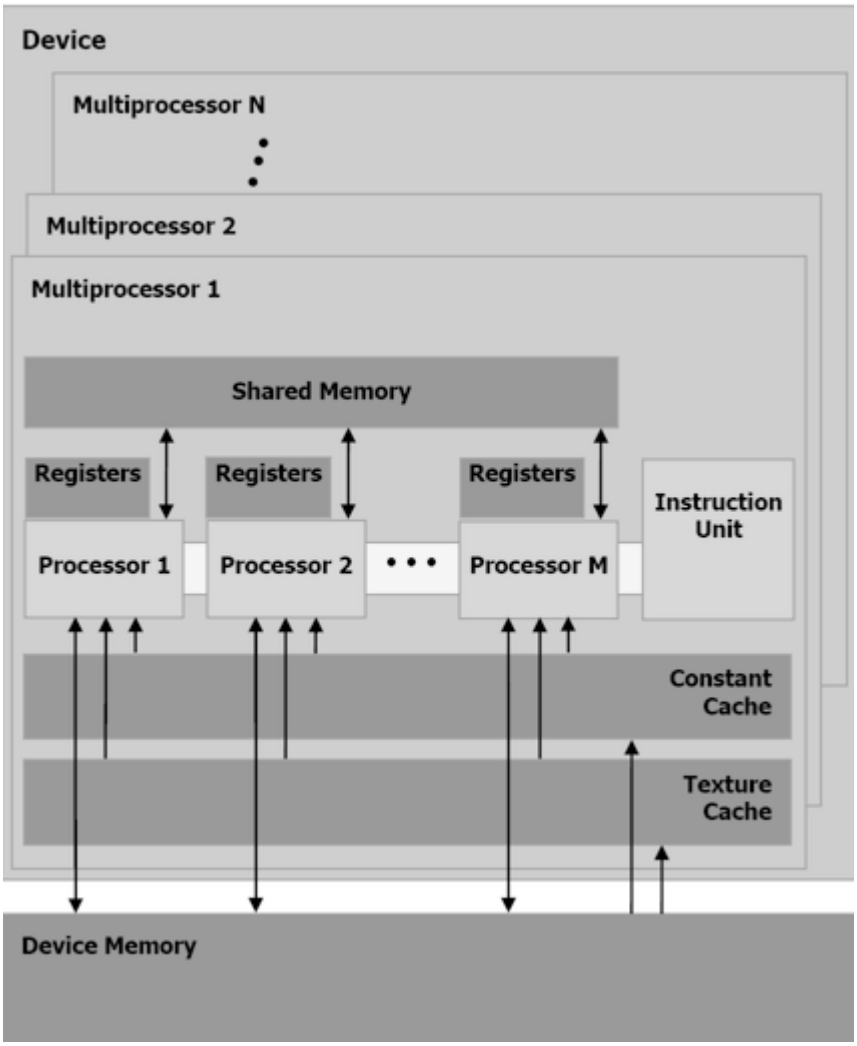


Figure 6.2: General hardware organization in the modern day GPU. Figure courtesy of the CUDA programming guide.

sorting. While on the CPU one is likely to use quicksort or mergesort, on the GPU the favourable algorithm is bitonic sort [4].

One of the most popular sub-areas in GPGPU revolves around the idea that most linear algebra operations can be performed much faster on the GPU than the CPU. CUDA even has a separate library called CUDA-BLAS which is meant to handle these operations. In the shader languages this has been handled by considering a matrix as a two dimensional texture. Kruger and Westermann [61] and Bolz *et al* [8] describe schemes of creating a library of linear algebra operations with particular emphasis on solving large scale sparse systems of equations using methods like Conjugate gradients. Our implementation is along very similar lines except that we use the new CUDA language as opposed to a shader language. Also, since we are interested in the specific kinds of matrices obtained during the course of solving graph cuts on grid graphs, we are able to write far more specialized procedures for certain operations. Goodnight *et al* [35] use the multigrid method on the GPU to solve boundary value problems similar to the Poisson equation we discussed. This is a hierarchical approach to solving BVPs.

In the field of Computer Vision the GPU has been used quite successfully for a variety of traditional problems. One of the interesting projects is OpenVIDIA [28] which aims to be the equivalent of OpenCV on the GPU. Stereo has been done using dynamic programming in the work of Gong and Yang [34] and by using belief propagation in Brunton *et al* [15] which implements the algorithm of Felzenzswalb and Huttenlocher [26]. Background subtraction has been tackled on the GPU using an extended collinearity criterion in the work by Griesser *et al* [39]. This algorithm manages to do background subtraction at a fast rate of 4ms per frame in an image sequence.

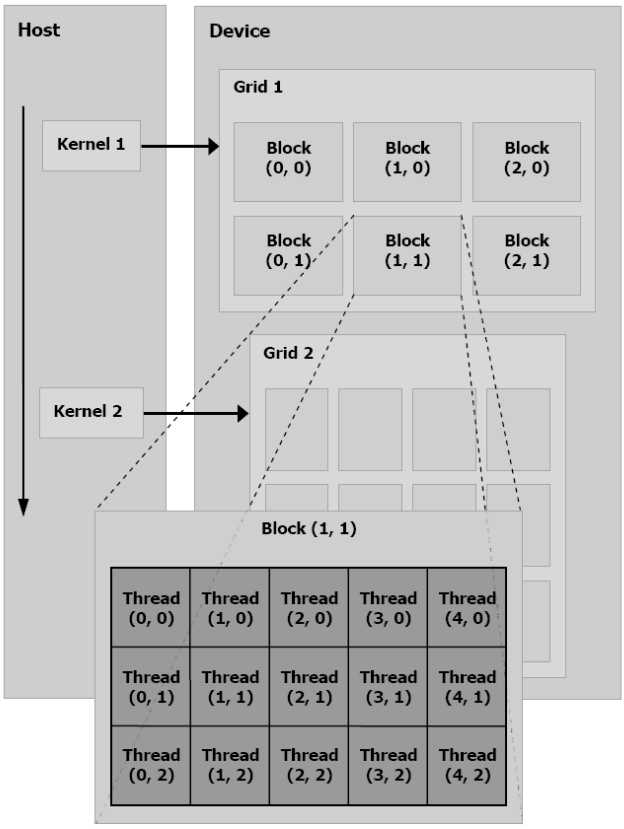


Figure 6.3: Illustration of the multi-threaded organization inside a CUDA kernel. Figure courtesy of the CUDA Programming Guide.

6.2.2 The 8800 and CUDA

CUDA enables the GPU to be programmed as a co-processor using a multi-threaded approach. Both the CPU and the GPU maintain their own DRAM and one can copy data from one to the other through optimized memcopy calls. Those instructions that fit into the SIMD model get off-loaded onto the GPU. A portion of the application that gets compiled into the instruction set of the GPU is called a *kernel* .

The threads that execute a kernel are organized into a grid of thread blocks as shown in figure 6.3.

The memory model for this multi-threaded execution is shown in figure 6.4.

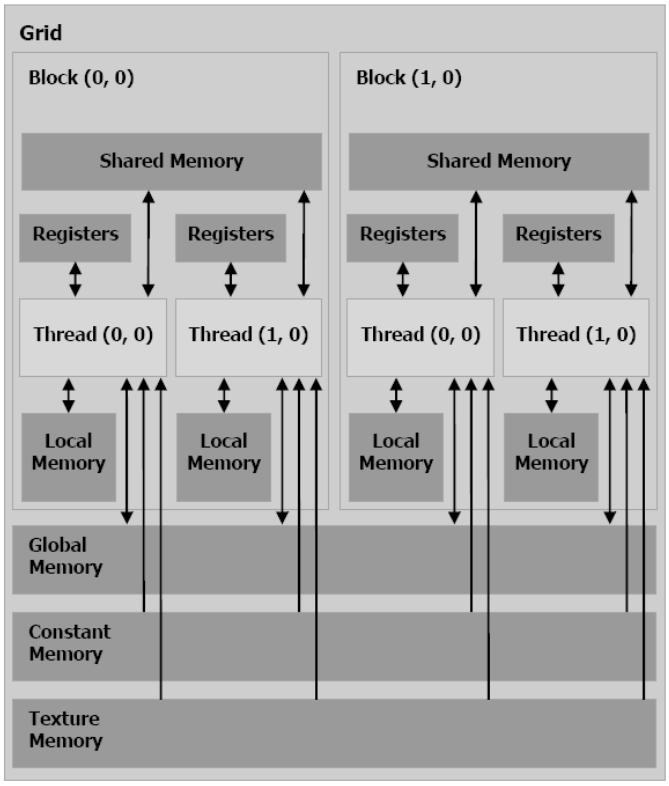


Figure 6.4: Illustration of the underlying memory model in CUDA. Figure courtesy of the CUDA Programming Guide.

Threads within a block can cooperate by accessing the shared memory. To synchronize threads within a block, synchronization points are specified by the *syncthreads()* command. Threads in a block are suspended until all of them reach the synchronization point. In the GeForce 8800 the maximum number of threads per block is 512. However, the blocks are combined to form a grid as shown in the figure and the maximum size of each dimension of the grid is much larger, 65535. This comes at the cost of decreased thread cooperation since threads in different blocks cannot communicate with each other. Hence, performance is affected by the choice of grid and block sizes and the call to the kernel from the host has to specify both these parameters.

In terms of the actual hardware the 8800 has 16 multiprocessors, each consisting of 8 processors. A block is processed by only one multiprocessor and the shared memory space is on chip and can be accessed very fast. While executing a kernel, each block is split into SIMD groups of threads called warps each of which contains the same number of threads. This warp size for the 8800 is 32. This warp is then executed in a SIMD fashion by the multiprocessor.

The amount of shared memory available per multiprocessor is 16KB and the total DRAM available is 768MB. The accesses to device memory are the expensive operations and take about 200 to 300 clock cycles to execute.

6.2.3 CUDA implementation

Our implementation in CUDA basically involves breaking the algorithm down into a few kernels. For ease of reference we again list the main algorithm along with the conjugate gradient subroutine.

As stated before, the basic operations that we need are SAXPY, pointwise multiplication, computing the ℓ_1 norm of a vector, computing the minimum of a vector and finally the matrix vector products ADA^Tx, Ag and A^Tx. The bottleneck step,

Algorithm 9 Calculate min-cut: $\min_{\nu} \|\text{diag}(\mathbf{w})(A^T\nu - \mathbf{c})\|_1$

choose t,μ
$\nu_i = s_i/(s_i + t_i)$
choose y such that $y \geq |A^T\nu - c|$
while change in ℓ_1 norm since last (outer) iteration above threshold$_1$ **do**
  while change in ℓ_1 norm since last (inner) iteration above threshold$_2$ **do**
    $z = (A^T\nu - c)$.
    Compute $\mathbf{d}$ from $d_i = 2/(y_i + z_i)^2$.
    Compute $\mathbf{g}$ from $\mathbf{g}_i = \frac{2\mathbf{z}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2} + \frac{2\mathbf{y}_i\mathbf{z}_i}{\mathbf{y}_i^2 + \mathbf{z}_i^2}(t\mathbf{w}_i - \frac{2\mathbf{y}_i}{\mathbf{y}_i^2 - \mathbf{z}_i^2})$
    Solve $(A\text{diag}(\mathbf{d})A^T)\Delta\nu = -A\mathbf{g}$ to get $\Delta\nu$ using Algorithm 10.
    Compute $\Delta\mathbf{y}$ from $\Delta y_i = \frac{y_i^2 - z_i^2}{2(y_i^2 + z_i^2)}\left(-tw_i + \frac{2y_i}{y_i^2 - z_i^2} + \frac{4y_i z_i (A^T\Delta\nu)_i}{(y_i^2 - z_i^2)^2}\right)$
    If necessary, scale step by β so that $\nu + \beta\Delta\nu$, $y + \beta\Delta\mathbf{y}$ are feasible.
  end while
  $t = \mu * t$
end while

Algorithm 10 Solve $A\mathbf{x} = \mathbf{b}$ using Conjugate Gradients

$\mathbf{x} = \mathbf{x_0}$
$\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x_0}$
$\mathbf{p} = \mathbf{r}$
$\rho = \mathbf{r^T r}$
while $\sqrt{\rho}/||\mathbf{b}|| > \epsilon$ **do**
  $\mathbf{a} = \mathbf{A}\mathbf{p}$
  $\alpha = \rho/(\mathbf{a^T p})$
  $\mathbf{x} = \mathbf{x} + \alpha\mathbf{p}$
  $\mathbf{r} = \mathbf{r} - \alpha\mathbf{a}$
  $\rho_{\text{new}} = \mathbf{r^T r}$
  $\mathbf{p} = \mathbf{r} + (\rho_{\text{new}}/\rho)\mathbf{p}$
  $\rho = \rho_{\text{new}}$
end while

as mentioned before, is the conjugate gradient solution to the sparse system of equations. Our current implementation does a single conjugate gradient iteration on an equation system involving $512 \times 512 = 262144$ variables in 0.65 ms. In the following we enumerate each of these kernels and point out which line numbers in the algorithms require their usage. We also specify the grid and block sizes that we use for our computation on images of size 512×512.

1. The matrix addition, pointwise multiplication and pointwise division operations that we see in lines 7,8 and 10 of Algorithm 9 are trivial kernels which we perform by having each thread compute one element of the result. There is one block for every row of the matrix and within each block there is one thread for every column i.e. 512 blocks each containing 512 threads. The 3 SAXPY operations in lines 8,9 and 11 of Algorithm 10 are done in exactly the same manner. The kernel currently computes a single such operation for 512x512 matrices in about 88 μs.

2. The inner product operations that take place in lines 7 and 10 of Algorithm 10 involve a pointwise multiplication followed by a summation of all the elements. In the CUDA implementation, if we want to sum up all the elements of a matrix A we divide A up into a certain number of blocks. The kernel is divided into blocks such that each block is responsible for taking one of these matrix blocks and producing a single number which is the sum of all the elements of the matrix block. Within each kernel block, we have the number of threads made equal to the block width. Each of the threads firstly sums up all of the elements down its column and then the threads cooperate among each other to find the sum of these partial sums. For a 512x512 matrix we found dividing it into blocks of 16x256 gave us the best speed. Summing up all the results produced by the threads is done through a process of reduction as shown in Algorithm 8 in the previous section. **partialsums** will be used as a shared array and the

threads within a block will be made to synchronize at step 8 of the algorithm using *syncthreads()*.

The process of computing the ℓ_1 norm of a vector, is similar, with the absolute values of the elements being added. This operation is needed in lines 4 and 5 of Algorithm 9. While computing β, the step size, in line 11 of Algorithm 9, it becomes necessary to compute the minimum element of a vector. For finding the maximum or the minimum element of a matrix, the summation operation in the procedure described above is just replaced by a comparison operation. The kernel for the inner product operation currently computes the dot product of two 512x512 length vectors in about 107 μs.

3. One of the crucial operations is the matrix-vector multiplication that is performed within the conjugate gradient iteration (line 6 of Algorithm 10). Since this operation is done on an average a 100 times, it becomes important to speed it up. Currently our kernel computes the product for a 512x512 graph in about 185 μs. Essentially all that needs to be done is shown in equation 6.1. To do this we employ a kernel that has each block computing a certain number of rows of the result. The number of threads within each block is the same as the number of columns in the matrix. Since the computation of each element basically involves access to its 4 neighbours, there is information that is read in during one element's computation that can be used for it's horizontal neighbour's computation. In the kernel therefore we load an entire row of p into shared memory. In this manner, the caching effect of shared memory can be used to speed things up.

   In our implementation we have 512 threads per block and we make each block compute 16 rows, which means that we have a total of 512/16=32 blocks.

4. We need to have a kernel to compute the matrix vector product Ag, which is needed to compute the right hand side vector of the normal equations in line 9

of Algorithm 9. A is the node-edge adjacency matrix and g is a vector of length m. This can be computed by simply accessing the correct elements. g can be divided into 4 parts corresponding to the s-links(g_s), t-links(g_t), horizontal(g_h) and vertical links(g_v). Then each element of the result is computed by the following basic computation.

$$result[i] = g_s[i] - g_t[i] + g_h[i-1] - g_h[i] + g_v[i-W] - g_v[i] \tag{6.2}$$

where W is the width of the image.

Here we make each thread compute one element of the result by having a block for each row and 512 threads for each block.

5. The matrix vector products $A^T * \nu$ and $A^T \Delta\nu$ being computed in lines 3,6 and 10 of Algorithm 9 are computed by having one thread for each element of x. Now A^T can be divided into 4 parts corresponding to the s-links, the τ-links, the horizontal links and the vertical links respectively. Each thread computes 4 entries, one corresponding to each of these four parts. The entries corresponding to the s and t parts are nothing but multiplication by the identity matrix and the negative identity matrix and hence are computed trivially. For the entries corresponding to the horizontal part we simply compute the difference of the current entry and its horizontal neighbour. Similarly for the ones corresponding to the vertical part, it is just the difference of the current entry and its vertical neighbour.

   Again this kernel uses a block for each row and 512 threads for each block.

Chapter 7

Experimental Results using the new Graph Cuts algorithm

This chapter shows results obtained using the new formulation of graph cuts on experiments 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 min-cut problem to the problem of solving a sequence of sparse linear systems, one can gauge the computational effort required to solve a given graph cut problem 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 conjugate gradient method were used in these experiments, the first variant employed the method without any preconditioning, the second made use of the diagonal preconditioner (also known as the Jacobi preconditioner) described in section 5.2.4 while the third used the preconditioning strategy described by Concus, Golub and Meurant [19].

Also, the actual time taken by the CUDA implementation on the NVIDIA 8800 GPU is presented. Comparisons with Kolmogorov and Zabih's implementation of flow-based min cuts are also provided. While the two algorithms use two different hardware components, we ran both on the same machine to offer some basis of

comparison. The CUDA implementation execution times also compare favourably with Dixit *et al* [23] who implemented the push-relabel algorithm on the GPU.

The proposed method was used for the purposes of image restoration and for interactive foreground-background segmentation.

7.1 Image Restoration

The algorithm was applied to the image restoration problem described by Boykov *et al* in [12]. The graph is formed using the energy function defined in Section 1.1. To reiterate, the data term is given by $D_p(L_p) = \min(|L_p - I_p|^2, const)$ and the smoothness term is given by $V(L_p, L_q) = k * \min(k1, |L_p - L_q|)$, where I_p is the intensity value at pixel p in the input image and L_p is the intensity value in the proposed restoration result.

In the synthetic example shown in Figure 7.1 a black and white image is corrupted by Gaussian noise. The proposed scheme required 237 CG iterations with no preconditioner, 156 with the Jacobi preconditioner and 32 with the Concus,Golub and Meurant preconditioner. The method took about 45ms using CUDA on the GPU for an implementation that uses the Jacobi preconditioner.

7.2 Interactive foreground-background segmentation

In the interactive segmentation problem, the user provides some level of input for the separation of foreground and background. While using energy minimization, the data term is based upon the user input and the smoothness term tries to ensure that two neighbouring pixels will only be put into different segments when there is a significant intensity difference between them. This model has been used by Boykov and Jolly [10] and Rother *et al* . [78]. Boykov and Jolly use a data term that is based

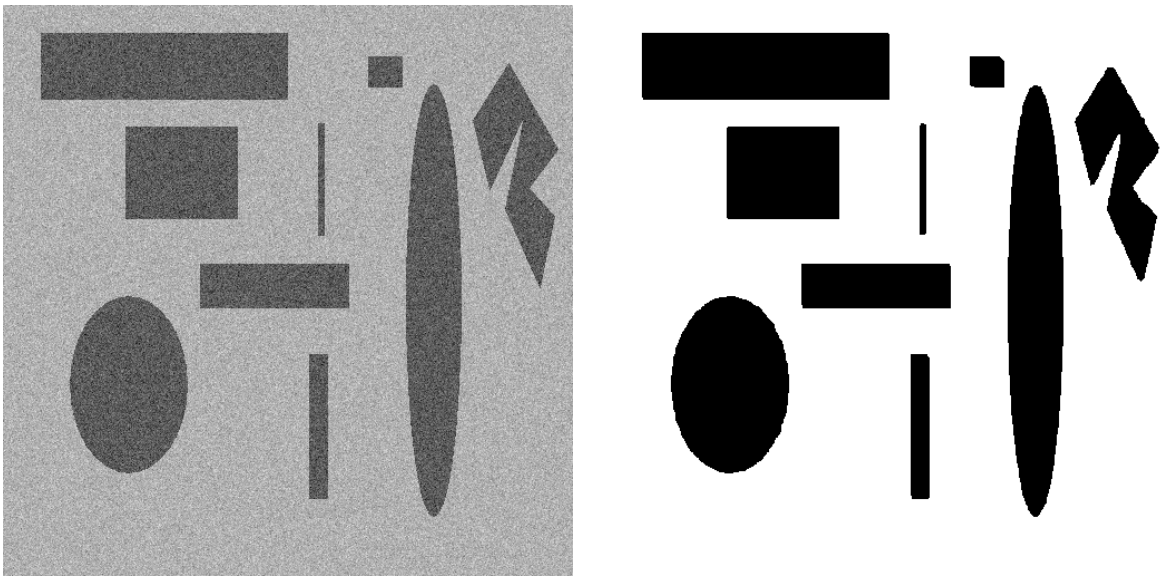


Figure 7.1: Example of image restoration using the proposed scheme.

on the histogram of gray level intensities sampled with the regions marked by the user. Rother *et al* , in a method they call Grabcuts, use Gaussian Mixture Modeling on the colours sampled with the user demarcated region. The smoothness term is just an exponentially decaying function of intensity difference. Methods for learning the parameters involved in this algorithm from image data have been investigated by Blake *et al* [7]

Grady and Funka-Lea [36] use a random walker approach for the same problem. In this approach a random walker starts off from a pixel that is not labeled by the user and takes on the label of the first labeled pixel that it reaches. An interesting aspect about this approach is that it ends up with equations involving the Laplacian of the underlying graph, very similar to the system of equations that arise in the unconstrained ℓ_1 norm approach proposed in this thesis.

In the system implemented in the thesis, given an image, the user initially marks out some areas as foreground and background. Using the marked pixels, just like Grabcuts, the parameters of two Gaussian Mixture Models are learnt, one each for the foreground and background. A mixture of 5 Gaussians is used in both cases.

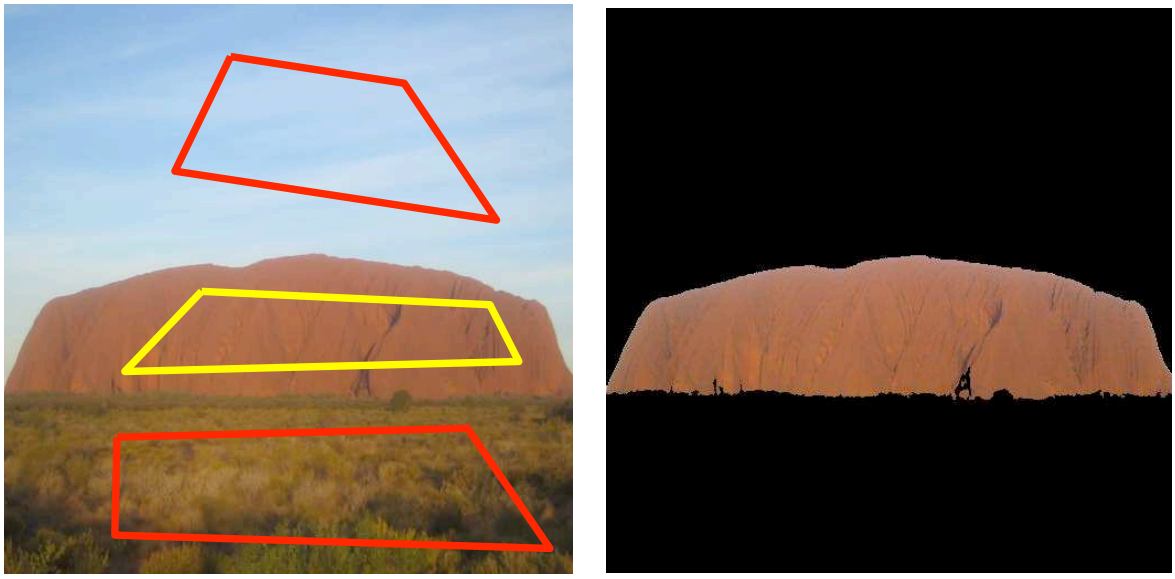


Figure 7.2: Ayers rock(Foreground-Background segmentation)

Using these GMMs, the probability and hence the data term (negative log of the probability) for each pixel being assigned to foreground or background can be determined. The internal edge weights for each pair of neighbouring pixels are provided by an exponentially decaying function of pixel intensity difference. Only one iteration of the GrabCuts procedure is performed.

The method discussed in the thesis was applied to the interactive foreground/background segmentation problems shown in figures 7.2 through 7.9. All of the images in question are 512×512.

In the remaining part of this section various aspects of the algorithm are analyzed in greater detail.

Firstly experiments were conducted to determine the effects of preconditioning. Table 7.1 shows the number of conjugate gradient iterations taken by each of the three variants of preconditioning.

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

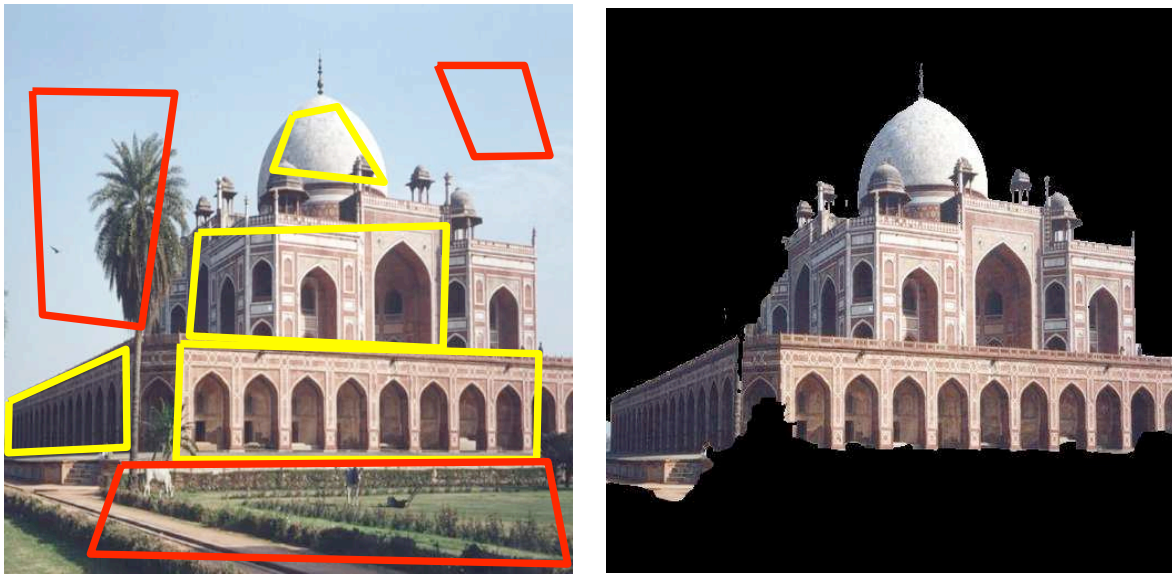


Figure 7.3: Humayun's tomb(Foreground-Background segmentation)

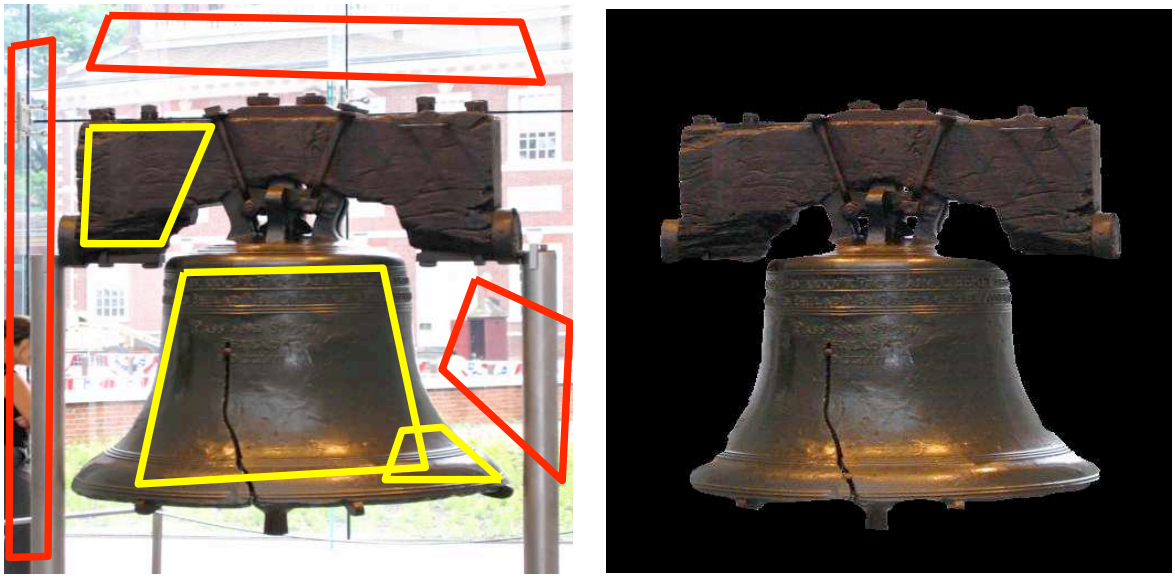


Figure 7.4: Liberty Bell(Foreground-Background segmentation)

Figure 7.5: Footballer(Foreground-Background segmentation)

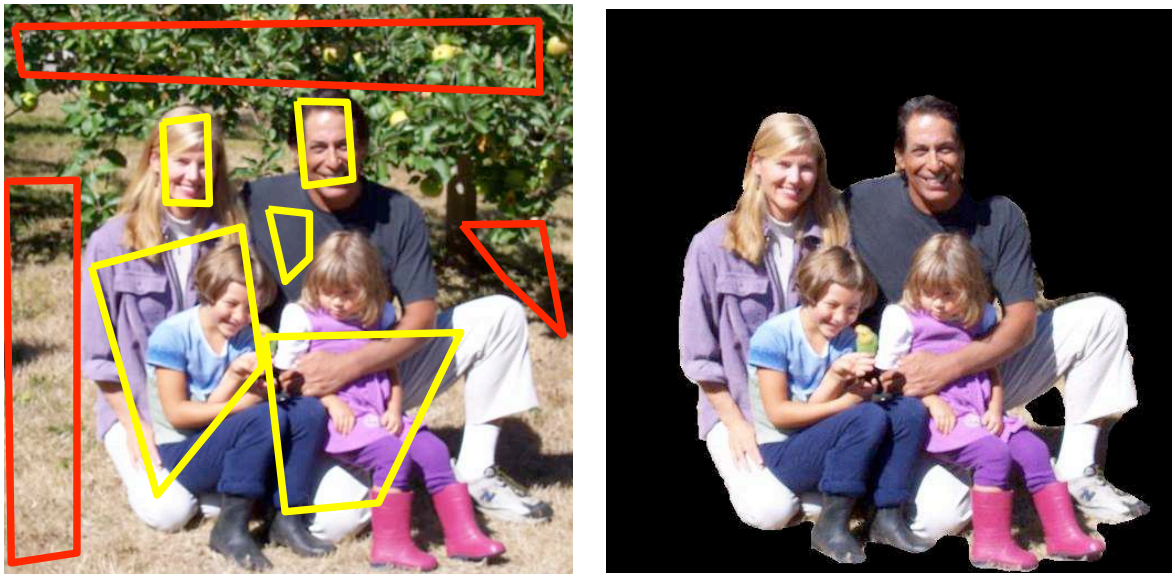


Figure 7.6: Family(Foreground-Background segmentation)

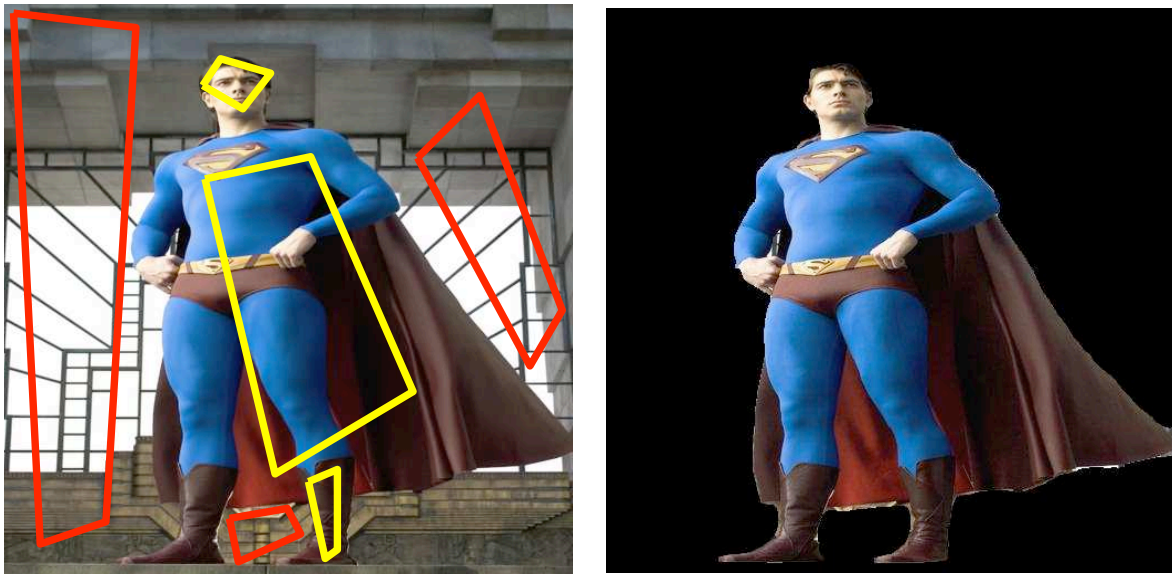


Figure 7.7: Superman(Foreground-Background segmentation)

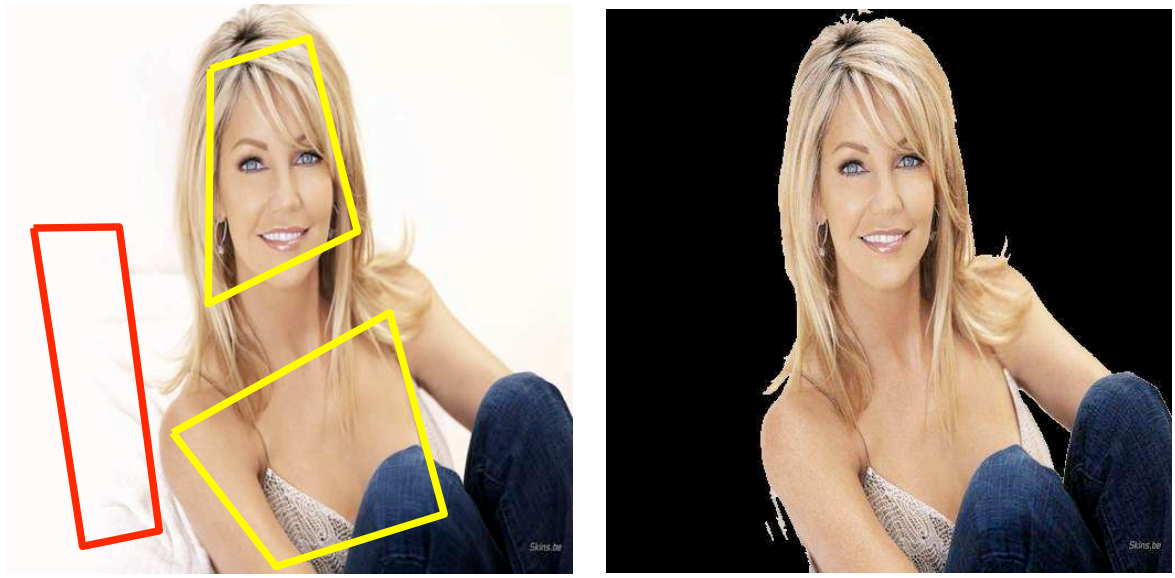


Figure 7.8: Actress(Foreground-Background segmentation)

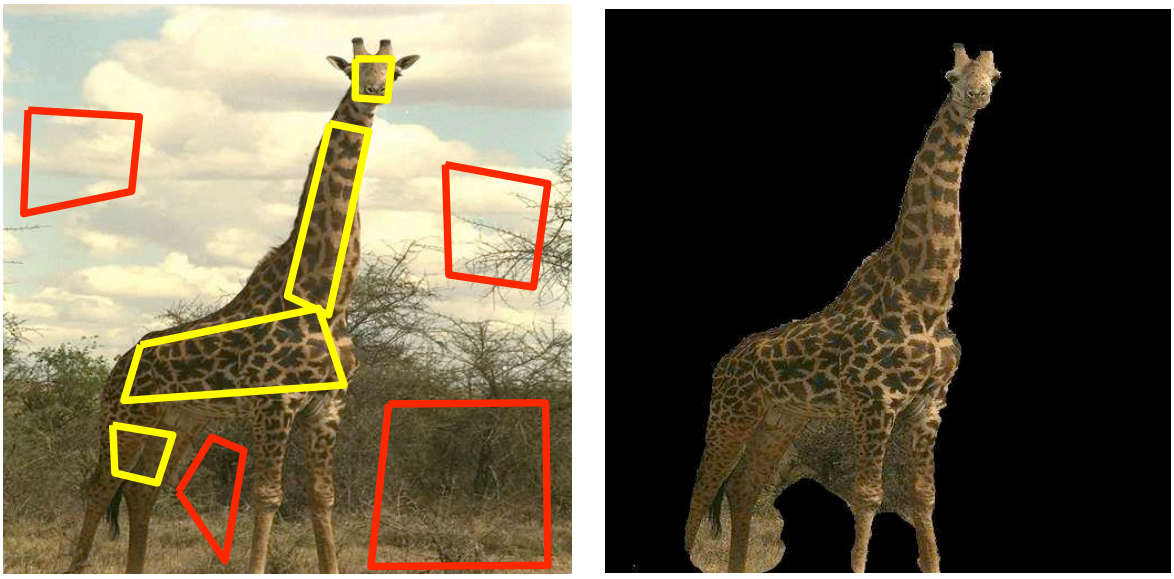


Figure 7.9: Giraffe(Foreground-Background segmentation)

Table 7.1: Number of iterations taken by the unbounded algorithm for separating the foreground in each image used in this thesis, using different preconditioning strategies

Image name	**CG iters**	**with Jacobi preconditioner**	**with preconditioner in [19]**
Ayers rock	411	74	19
Tomb	226	120	31
Liberty Bell	272	105	12
Footballer	379	90	25
Family	279	132	25
Superman	435	95	23
Actress	674	74	26
Giraffe	351	143	23

Table 7.2: Number of Newton steps taken for the algorithm to converge on various images

Image name	**Newton steps**
Ayers rock	9
Tomb	10
Liberty Bell	8
Footballer	11
Family	6
Superman	8
Actress	14
Giraffe	6

a factor of 0.27 on average while the Concus and Golub preconditioner reduces that number even further. The Concus and Golub precondtioner results in fewer iterations but each iteration takes more time. MATLAB implementations of both Jacobi preconditioners and the Concus and Golub variant indicate the Jacobi preconditioner to be about 100 times faster.

In every case the implementation converged to an acceptable solution in fewer than 15 newton steps as is shown in table 7.2. This is consistent with the kind of performance that interior point methods exhibit on a wide variety of linear programs.

Figure 7.10 gives a comparison of the energy obtained by the combinatorial graph cuts method and the energies obtained during the process of using the method at the end of each Newton step.

To analyze the computational effort involved in this algorithm, Table 7.3 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 7.4 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

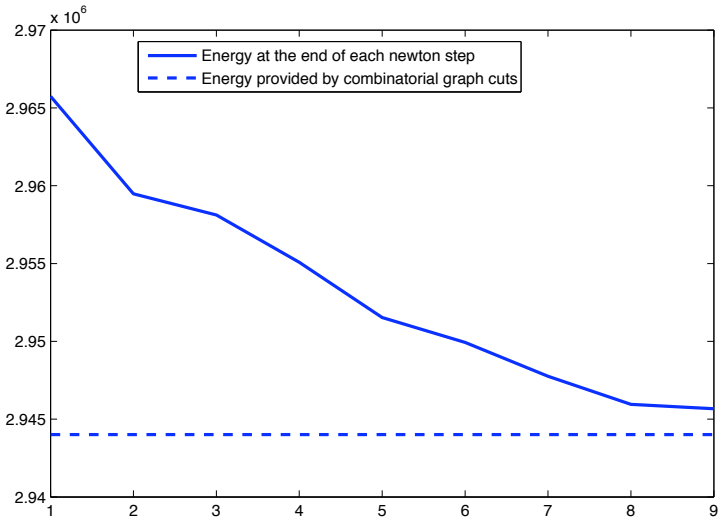


Figure 7.10: Energy value at the end of each Newton step. It can be seen that the energy converges towards the value obtained by combinatorial graph cuts.

Table 7.3: Total number of floating point operations needed for foreground extraction.

Image name	**Floating point operations** (x 10^9)
Ayers rock	0.74
Tomb	1.04
Liberty Bell	0.93
Footballer	0.87
Family	0.90
Superman	0.87
Actress	0.87
Giraffe	0.94

Table 7.4: Variation of the number of Conjugate Gradient iterations with image size.

Image size	**CG iters**
128x128	86
256x256	90
512x512	90
1024x1024	151

Table 7.5: Parameters used in the foreground-background experiments

Image name	t_0	μ	**threshold**$_1$	**threshold**$_2$
Ayers rock	1	2	1	1
Tomb	1	2	1	1
Liberty bell	1000	10	2	5
Footballer	1	2	1	1
Family	10	2	1	1
Superman	10	2	1	1
Actress	1	2	1	1
Giraffe	10	2	1	1

set of experiments. While the total number of conjugate gradient iterations seem to be about the same (apart from the 1024x1024 image), the computational effort required to perform a single conjugate gradient iteration will increase linearly with the number of pixels in the image as we will see in Section 7.2.1.

In table 7.5 we list the parameters used in our algorithm for these various images. These parameters were hand picked to get the least possible conjugate gradient iterations. The parameters to be chosen are the initial t value(t_0), μ and the two convergence thresholds for the inner and outer iterations in algorithm 9. The parameters are around the same for most of the images apart from the liberty bell image.

Figure 7.11 shows how the ℓ_1 norm reduces with the number of Newton steps for the superman image. It is almost always the case that the initial Newton steps move the solution close to the optimal solution and the last steps are responsible

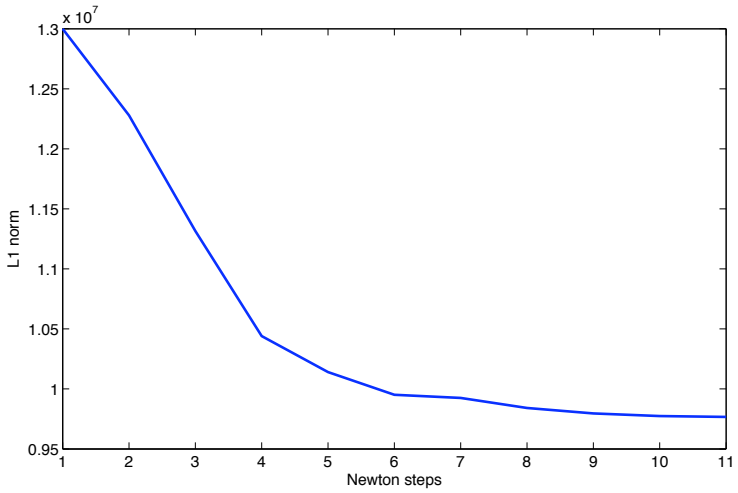


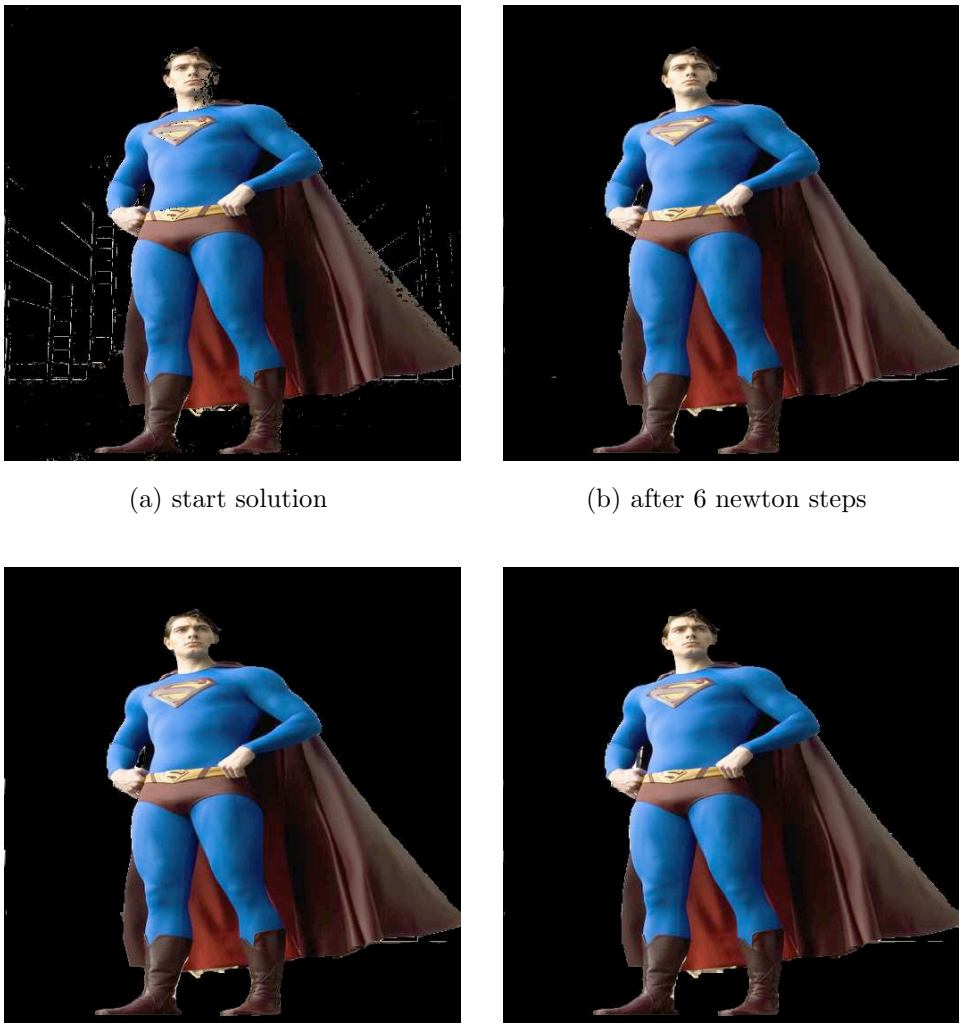
Figure 7.11: The decrease of the ℓ_1 norm with each Newton step(for the superman image). The initial steps result in a more dramatic decrease that the later steps.

for correcting fewer pixel labels (such as those around the foreground-background edges) and hence are not changing the ℓ_1 norm by much. Therefore in terms of the reduction of ℓ_1 norm one observes diminishing returns as the number of Newton steps increase.

These later Newton steps also usually take up more conjugate gradient iterations. Table 7.6 indicates this for the superman image.

In terms of the quality of the results, the initial Newton steps drive the solution close to the optimal and greater effort is spent around places such as the edges. To illustrate this, Figure 7.12 shows the results at different Newton steps for the superman image.

From Figure 7.12 and Table 7.6 one can see that the final 42 conjugate gradient iterations are not contributing much in terms of the quality of the end result. Therefore, one can imagine an early termination of the algorithm without significant loss in quality of the result.



(a) start solution

(b) after 6 newton steps

(c) after 7 newton steps

(d) final result

Figure 7.12: The various stages of the graph cuts algorithm on the superman image

Table 7.6: Number of conjugate gradient iterations for each Newton step in the superman image.

t	Newton step number	CG iters	Cumulative CG its
1	1	5	5
1	2	5	10
1	3	5	15
1	4	5	20
1	5	8	28
1	6	19	47
10	7	6	53
10	8	15	68
100	9	8	76
1000	10	19	95

7.2.1 Results from the CUDA implementation

The proposed unconstrained scheme from Algorithm 6 was implemented on an NVIDIA GeForce 8800 GTX GPU using NVIDIA's newly released Compute Unified Device Architecture (CUDA). The implementation was applied to the segmentation problems shown in Figures 7.2 through 7.9 and the timings achieved on these 512x512 problems are summarized in Table 7.7. For this implementation, the unbounded version of the algorithm was used with the Jacobi preconditioner to solve for the equation systems.

For purposes of comparison, we also applied the flow based graph cut method proposed by Boykov and Kolmogorov [11] to the same problems and recorded the timings achieved with a 2.66 GHz Intel Core 2 Duo processor with a 4MB cache. Although the two implementations use different hardware, these experiments provide some basis for comparison as they were performed on the same machine. It can be seen that both methods perform the segmentation in a small fraction of a second and in most cases take around the same time. The results obtained with the proposed interior point scheme also compare favourably to those obtained with the

Table 7.7: Time taken (in milliseconds) to extract the foreground of images on the GPU and using a flow based method on the CPU.

Image name	**Boykov (in ms)**	**CUDA (in ms)**
Ayers rock	60	62
Tomb	65	92
Liberty Bell	63	57
Footballer	62	58
Family	73	86
Superman	61	71
Actress	62	69
Giraffe	74	92

Table 7.8: Time taken for a single Conjugate gradient iteration on images of different sizes

Image size	**CG iteration time(ms)**
128x128	0.280
256x256	0.287
512x512	0.512
1024x1024	1.902

GPU scheme described in [23] which implements the push relabel method of computing maximum flow. Their work was done on an ATI Radeon 9800 Pro card and for the purposes of image denoising this implementation took around 3200 ms. It is worth noting that graphics cards have undergone changes since [23] was published, however we do not have any information on the running times on current hardware.

The bottleneck step is that of solving the system of normalization equations. The time taken to perform a single conjugate gradient iteration for images of different sizes is given in Table 7.8. 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 Jacobi preconditioning procedure.

7.2.2 Memory bound algorithms v/s compute bound algorithms

Interestingly the GPU version of this algorithm is currently memory bound rather than compute bound since this particular GPU offers a theoretical peak performance of 330 GFLOPS. Using values from tables 7.3 and 7.7 our algorithm currently is running only at 16 GFLOPS. It is expected that further restructuring of the GPU code to reduce memory traffic will significantly reduce the execution time. Similarly, as the proposed approach is implemented on systems with greater memory bandwidth, greater on-chip memory or larger numbers of processors its performance will scale linearly.

To get an idea of the speeds that can practically be achieved on this GPU consider the time taken by two simple operations, inner-products and SAXPY on 512×512 matrices. Saxpy is performed at 88 μs and inner-products at 107 μs. Saxpy involves 2 operations per element and hence the number of GFLOPS obtainable here equals $512 * 512 * 2/88 = 6$ GFLOPS. Therefore it can be seen how the latency between the GPU Device Memory and the processors affect the performance even in the case of simple operations. The ideal program therefore needs to pack a lot of arithmetic operations into each kernel so that the DRAM latency will be hidden by the computationally intensive operations being performed by the processors.

7.3 Potential further work on graph cuts

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

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. Dynamic MRFs are an important topic of research since they arise very naturally in problems such as video segmentation. Currently our algorithm does not have a principled way of taking an initial solution and getting an optimal solution in minimal time. Investigating this question should provide us with an answer to solving dynamic MRFs, since the change in the graph from frame to frame in a video sequence is not much and hence a good solution for the n^{th} should be a good solution for the $(n+1)^{\text{th}}$ frame.

Another topic that deserves further attention is the issue of preconditioning strategies. The experiments indicate that more sophisticated preconditioning strategies such as those described in [19] can dramatically reduce the number of conjugate gradient iterations required. As mentioned before, this particular preconditioner does not fully exploit the parallelism on the NVIDIA 8800 due to the structural limitations of this processor, however, the method is likely to work well on other parallel systems such as the Cell processor. Here one would need to weigh the time required to solve the preconditioner against the reduction in conjugate gradient iterations.

Chapter 8

Formulating Stereo Matching as a Linear Program

This chapter describes an approach to solving the stereo matching problem via a novel method. The method proceeds by constructing a convex approximation to the problem and solving this directly using interior point barrier method.

In particular through the convexification of the data term involved in stereo, a convex objective function is created which can then be minimized using interior point methods in a manner very similar to the graph cuts problem.

While the problem being tackled is stereo, the framework is general enough to be extended to problems like motion and determination of parameters for warping, which, like stereo are more suited to solution methodologies in a continuous domain.

This chapter introduces the stereo matching problem, discusses related work and then shows one how to convert the problem into a linear program. Chapter 9 shows how the interior point barrier method that was applied to the graph cuts problem can be readily applied to the stereo LP with very similar equations. Finally chapter 10 provides results on the standard Middlebury data set.

Figure 8.1: The Bumblebee stereohead from PointGrey

8.1 Introduction

Stereo matching is one of the classic problems in computer vision. It aims at finding pixel correspondences across a pair of images taken of the same scene from different locations. These two images are either obtained from a stereo head such as the Bumblebee shown in Figure 8.1, or by using a single camera and displacing it from one position to the other. In the most general case, the two positions could be arbitrary, but for the purposes of this thesis proposal we will be dealing with pairs of images from the Middlebury data set [81].

For any pair of images, it is possible to transform them so that the search for correspondences is reduced to a 1-D search problem along the horizontal raster lines of the images. This is known as rectification (this can be done quite compactly given the camera projection matrices. See for example [29]). The pairs in the Middlebury dataset are provided as rectified pairs.

Let us denote the left image as L and the right image as R. Then given a pixel location in the left image (x_L, y_L), the corresponding pixel in the right image is to

be found at some location (x_R, y_R), $x_R = x_L - d, y_R = y_L$ where d is the disparity. The aim is to accurately determine d for each pixel, thus creating what is called the disparity map $d(x, y)$.

Given two pixels that correspond, it is possible to triangulate the two matching pixels to determine the location of the point in 3D space that got imaged to these two pixels as described in most standard computer vision textbooks such as [41]. The main application of stereo therefore is to detect depth in the scene. This is used for example in obstacle avoidance [16].

Like many other problems in the field, the stereo problem is often rephrased as an optimization problem where the goal, in this case, is to minimize an objective function which includes some terms that model the costs associated with matching pixels at various disparities and others that seek to reward overall 'smoothness'.

These optimization problems are often solved using discrete optimization techniques such as belief propagation or graph cuts. This chapter seeks to demonstrate that continuous convex optimization techniques can also be effectively employed in this context. The original objective function is approximated by a convex variant which can be tackled using interior point methods. One can exploit the structure of the resulting problem to effectively minimize an objective function involving hundreds of thousands of continuous variables in a manner very similar to the graph cuts problem seen in the previous chapters.

The resulting approach has a number of useful features. Firstly it allows us to handle problems, like stereo, where the decision variables are continuous without requiring any intermediate quantization. Secondly, one can naturally incorporate penalty terms involving more complex functions of the disparity values. This flexibility is exploited by introducing a penalty term involving the Laplacian of the disparity values which allows us to better model slanting surfaces. Thirdly the approach allows the stereo problem to be reformulated as a sequence of convex optimization problems which have the usual useful properties, such as a guarantee that any local

minimum must be the global minimum.

8.2 Related Work

Scharstein and Szeliski have published an excellent taxonomy of stereo algorithms in [80]. In this taxonomy, solution schemes can be divided into local methods where individual pixels or pixel windows are matched across images and global methods where a global energy function needs to be minimized.

Yoon and Kweon [99] introduce a new similarity measure that is based upon the central assumption that a pixel in the left image and a pixel in the right image that look alike and happen to be distinctive in each image, are more likely to be a matched pair. Among local strategies this is the one that performs best.

Almost all of the top ranked algorithms on the Middlebury data set [81] are global methods. Optimization methods that have been successfully applied to this problem include dynamic programming, belief propagation and graph cuts.

The most successful dynamic programming method is that of Hirschmuller [43, 44]. In this work the energy function is optimized along 8 or 16 different directions in what he terms a Semi Global Matching(SGM) method. Cues from segmentation are used for handling untextured areas.

Many of the top ranked algorithms use belief propagation which is usually performed using some variant of the method described in [26]. Sun *et al* [87] use a symmetrical energy function which takes into account disparity maps as well as occlusion maps from both images. Klaus *et al* [52] use oversegmentation and robust plane fitting through the segments to come up with a set of candidate disparity planes. Belief propagation is then used to solve an energy function on the segments. An iterative algorithm using hierarchical belief propagation is proposed in Yang *et al* [95] where the data term of the energy function is re-weighted in regions which are occluded or unreliable.

Utilization of graph cuts as an optimization technique can be seen for example in Kolmogorov and Zabih [57].

To date linear programming has not been featured as a method for solving the kinds of optimization problems commonly encountered in this context.

While linear programming has not been successfully applied to stereo previously, it is worth noting that it has been applied to motion estimation [48, 5]. The work by Jiang *et al* . [48] on matching feature points is similar in that the data term associated with each pixel is approximated by a convex combination of points on the lower convex hull of the match cost surface. However, their approach is formulated as an optimization over the interpolating coefficients associated with these convex hull points which is quite different from the approach described in this paper. Also their method uses the simplex method for solving the LP while the approach described in this paper employs an interior point solver which allows us to exploit the structure of the problem more effectively.

The remainder of this chapter describes how solving the stereo matching problem can be approximated as the solution to a huge linear program.

8.3 Formulation of the linear program

In this section the problem formulation will be presented in a 1 dimensional setting (per scanline) for ease of explication. This formulation can easily be extended to two dimensions by including both horizontal and vertical smoothing terms. All of the results described in Chapter 10 were obtained by optimizing just such a 'two dimensional' global objective function over all of the pixel values.

The disparities along a scanline are represented by a vector $\mathbf{d} \in \mathbb{R}^{\mathbf{n}}$ where d_i denotes the disparity associated with pixel i as shown in Figure 8.2. As is the case with most energy minimization formulations, the objective function is expressed as the sum of a data term, E_{data}, and a smoothness term. The smoothness term is in

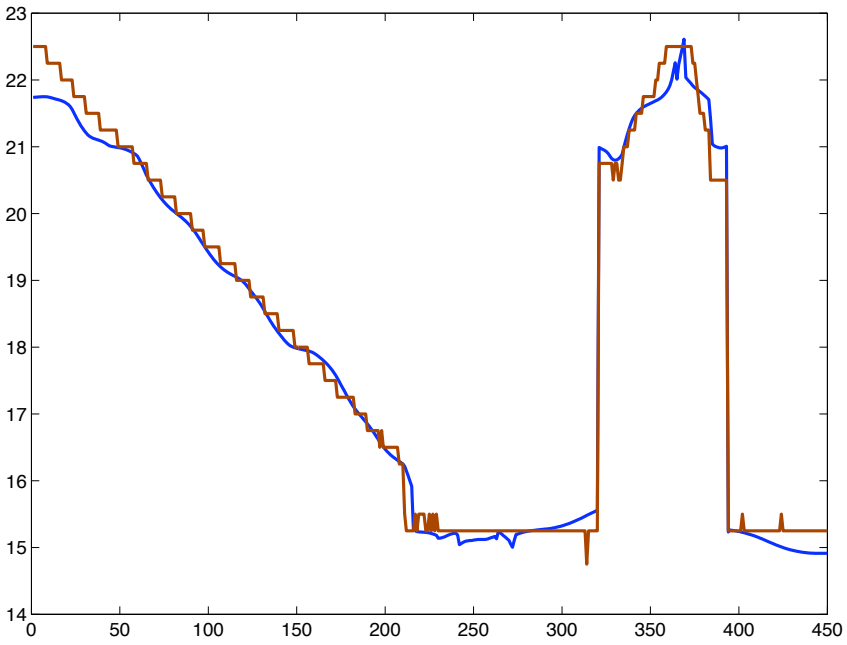


Figure 8.2: Solution of stereo matching along one scanline of the teddy image. The dashed line shows ground truth disparities and the solid line shows the solution obtained by the interior point method discussed in this thesis.

turn divided into two components, E_{grad} and E_{lap} which penalize the first and second derivatives of the disparity function respectively.

$$E(d) = \sum_{i=1}^{W} E_{\text{data}}(d_i) + \sum_{i=1}^{W-1} E_{\text{grad}}(d_i, d_{i+1}) + \sum_{i=2}^{W-1} E_{\text{lap}}(d_{i-1}, d_i, d_{i+1}) \tag{8.1}$$

The incorporation of a penalty on the second derivative allows us to express a preference for piecewise linear disparity solutions which allow us to model slanted surfaces in the scene. The addition of this Laplacian term in the energy function is a significant difference from most energy minimization schemes which typically consider only pairwise interactions between the pixels.

For the gradient based energy term a natural choice is simply the absolute difference $|d_i - d_{i+1}|$, but to allow for small changes in disparity we choose

$$E_{\text{grad}}(d_i, d_{i+1}) = \begin{cases} 0 & \text{if } |d_i - d_{i+1}| \leq \epsilon \\ w_{g_i}(|d_i - d_{i+1}| - \epsilon) & \text{otherwise} \end{cases} \tag{8.2}$$

Where w_{g_i} is a non-negative weight associated with the gradient at pixel i.

For the Laplacian based energy term we just use the discrete approximation of the Laplacian.

$$E_{\text{lap}}(d_{i-1}, d_i, d_{i+1}) = w_{l_i}|d_{i-1} - 2d_i - d_{i+1}| \tag{8.3}$$

Again in a manner similar to the gradient term, w_{l_i} represents the weight associated with the Laplacian at pixel i.

These absolute value terms can be captured with linear inequalities by introducing auxiliary variables, y_{g_i} and y_{l_i}, which serve as proxies for these terms as described in [9]. To capture the effect of $E_{\text{grad}}(d_i, d_{i+1})$ we introduce the following constraints

$$
\begin{aligned}
y_{g_i} &\geq d_i - d_{i+1} - \epsilon \\
y_{g_i} &\geq d_{i+1} - d_i - \epsilon \\
0 \leq y_{g_i} &\leq g_{\text{Max}}
\end{aligned} \tag{8.4}
$$

Similarly the variables y_l capture the effect of $E_{\text{lap}}(d_{i-1}, d_i, d_{i+1})$

$$
\begin{aligned}
y_{l_i} &\geq -d_{i-1} + 2d_i - d_{i+1} \\
y_{l_i} &\geq d_{i-1} - 2d_i + d_{i+1} \\
0 \leq y_{l_i} &\leq l_{\text{Max}}
\end{aligned} \tag{8.5}
$$

The parameters g_{Max} and l_{Max} represent upper bounds on the values for the gradient and Laplacian terms respectively. They can be derived from the range of legal disparity values.

At this stage, the only component of the energy function that is not convex is the data term. We address this issue in the following subsection.

8.3.1 Convex approximation of the data term

Conventionally in stereo the data term associated with assigning the label d to a pixel (x_L, y_L) is given by some function of the intensity differences between a patch around $L(x_L, y_L)$ and a patch around $R(x_L - d, y_L)$. Assume we have a discrete set of potential disparity values $d_{\text{min}} \leq \ldots \leq d_{\text{max}}$. Corresponding to each disparity value we obtain a score that indicates the cost associated with that correspondence. Given these scores, the disparity value that should be chosen based only on this information is the one that provides the minimum score. Our aim is to lower bound this score function with a continuous, convex function of the disparity.

To do this we construct a piecewise linear function in a manner similar to computing the convex hull of the entries in the score array as shown in Figure 8.3. In

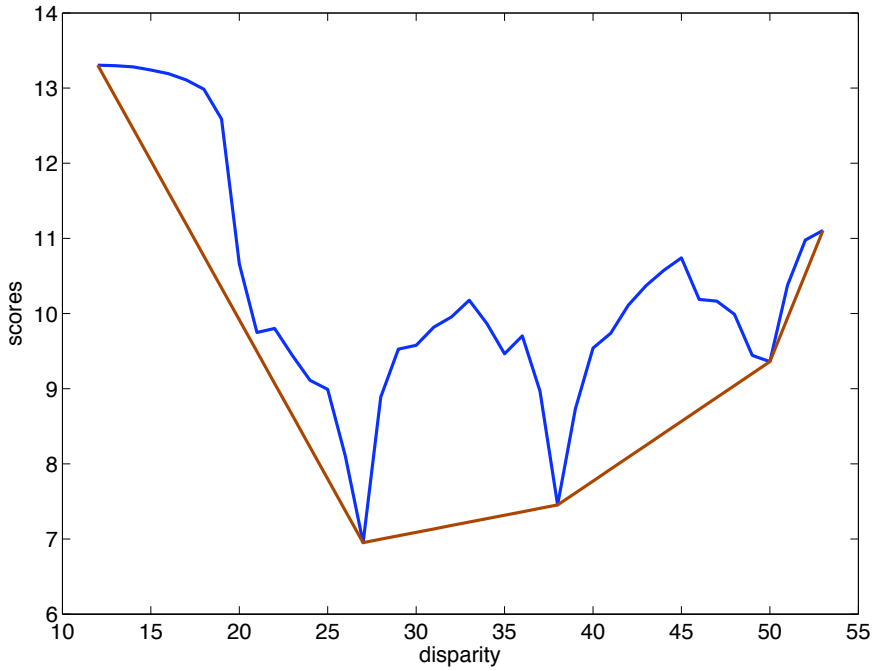


Figure 8.3: The score function associated with one of the pixels in the Teddy data set along with the corresponding piecewise linear convex approximation. In this case the lower approximation captures the ambiguity associated with the multiple matches. This ambiguity is later resolved because of the contributions of the smoothness terms.

situations where the pixel under consideration has a single well defined match in the other image the convex approximation will capture that fact. In situations where the pixel is not unique or where it has multiple possible matches the approximation will reflect that ambiguity which will ultimately be resolved by considering the smoothness terms.

Note that we do not require the disparity values in the (disparity, score) pairs to correspond to integral values. This allows us to easily incorporate subpixel disparity scores directly in the score function.

This convex lower bound function associated with pixel i is modeled by introducing yet another auxiliary variable y_{m_i} which is constrained to have a value greater than all of the linear segments that comprise the lower bound. More specifically, if

there are S_i segments in the approximation of the score function for the i^{th} pixel, then we have the following constraints corresponding to the match variables.

$$\begin{aligned} y_{m_i} &\geq a_{ji}d_i + b_{ji},\ j \in \{1, 2, \ldots S_i\} \\ 0 &\leq y_{m_i} \leq y_{Max_i} \end{aligned} \tag{8.6}$$

where variables a_{ji} and b_{ji} represent the slope and intercept for the j^{th} segment. The data term E_{data} can now be written as a weighted sum of these y_m score values.

Therefore the final formulation of the objective function can be written as follows:

$$E(d) = \sum_{i=1}^{W} w_{m_i} y_{m_i} + \sum_{i=1}^{W-1} w_{g_i} y_{g_i} + \sum_{i=1}^{W-2} w_{l_i} y_{l_i} \tag{8.7}$$

This objective function is then optimized under the matching constraints of 8.6, the gradient constraints of 8.4 and the Laplacian constraints of 8.5.

We would, ideally, like to arrange matters such that the weights associated with the variables, w_{m_i}, w_{g_i}, and w_{l_i}, reflect the occlusion structure of the scene. More specifically, for pixels that are half occluded the associated w_{m_i} values should be set to low values so that their match score is ignored in the overall optimization. Similarly the weights associated with the gradient and Laplacian term should be large everywhere except at occlusion boundaries and places where the disparity gradient changes discontinuously. Section 9.1 discusses how these weights are estimated from the available data.

8.4 ℓ_1 versus ℓ_2

It is important, especially in the context of stereo to understand why the ℓ_1 norm is advantageous over the ℓ_2 norm when it comes to penalizing changes in disparity.

Assume optimization is being performed only over a scan-line, that is, consider the problem to be one dimensional for the time being. We have 10 pixels $x_1, \ldots, x_{10}$

Table 8.1: Filling in unknown disparity values based on information from the 'reliable' pixels.

Pixel no:	1	2	3	4	5	6	7	8	9	10
Disparity	70	70	70	70	70	?	?	?	80	80

Table 8.2: Solution 1 to the problem in table 8.1 of filling in unknown disparities - An ℓ_1 norm based solution.

Pixel no:	1	2	3	4	5	6	7	8	9	10
Disparity	70	70	70	70	70	70	70	70	80	80

whose disparities have to be determined. Say we reliably determine that the disparity of $x_1, \ldots, x_5$ is 70 and the disparity of x_9 and x_{10} is 80 as shown in table 8.1.

Now assuming that all the match weights and gradient weights are equal, consider the two potential solutions in Tables 8.2 and 8.3.

If an ℓ_2 norm is used to penalize differences between neighbouring disparities, the first solution will give rise to a penalty of 100. The second solution will result in a penalty of only $3 \times (2.5)^2 = 18.75$ and hence will be preferred. However, the actual solution is more likely to be the one with the sudden jump in disparity as opposed to the gradual disparity increase since it is likely that the disparities of 70 correspond to one object and the disparities of 80 correspond to another object and therefore there will be a sudden jump as we transition from one object to the other along the scanline.

On the other hand, if the ℓ_1 norm is used, then both solutions will be penalized

Table 8.3: Solution 2 to the problem in table 8.1 of filling in unknown disparities - An ℓ_2 norm based solution.

Pixel no:	1	2	3	4	5	6	7	8	9	10
Disparity	70	70	70	70	70	72.5	75	77.5	80	80

equally. In this case, both will pay a penalty of 10. Therefore, unlike the ℓ_2 norm there is no reason for the optimal solution to be biased towards gradual changes in disparity over sharp changes in disparity. Since the sharp changes in disparity are a more commonly occurring phenomena, usage of the ℓ_1 norm proves to be advantageous.

The robustness of the ℓ_1 norm over the ℓ_2 norm is a well known property that has been used in the domain of motion analysis in work by Ben-Ezra *et al* [5] which uses the ℓ_1 norm as a measure of residual error for point-line correspondences. Again, linear programming is used to solve this problem. L^1 minimization has been used to solve stereo problems using a PDE approach in work by Kumar *et al* [62]. The smoothness term in the energy formulation for optical flow estimation in recent work by Brox *et al* [14] makes use of a modified ℓ_1 minimization to robustly account for discontinuities at object boundaries in the scene.

Chapter 9

Solution of the stereo LP using barrier method

This chapter describes how one can use the interior point barrier method described in Chapter 4 to solve the stereo LP described in the previous chapter. Section 9.1 details the equations that come up as one uses this method and the second section 9.2 shows the kind of decisions that need to be made while solving stereo.

9.1 Using interior point method to solve the LP

The linear programming problem given in Equation 8.7 can be stated more compactly as follows:

$$\begin{aligned} &\min w^T x \\ \text{st} \quad &Ax \leq b \end{aligned} \tag{9.1}$$

where x is a vector formed by concatenating all of the decision variables, $x = \begin{bmatrix} d & y_m & y_g & y_l \end{bmatrix}^T$, and w is the vector formed by the corresponding weights, $w = \begin{bmatrix} 0 & w_m & w_g & w_l \end{bmatrix}^T$.

b is a column vector that concatenates the intercepts of the line segments used for the convexification, the bounds on disparities and the bounds on Laplacian and gradient terms.

The matrix A concatenates all of the linear constraints and is given by

$$A = \begin{bmatrix} A_m & -I_m & 0 & 0 \\ B_d & 0 & 0 & 0 \\ 0 & B_m & 0 & 0 \\ G & 0 & -I & 0 \\ -G & 0 & -I & 0 \\ 0 & 0 & B_g & 0 \\ L & 0 & 0 & -I \\ -L & 0 & 0 & -I \\ 0 & 0 & 0 & B_l \end{bmatrix} \tag{9.2}$$

where the B_d, B_m, B_g, B_l matrices correspond to the bounds on the d, y_m, y_g, y_l variables respectively.

The A_m and I_m matrices help to capture the constraints associated with the convex lower bounds. A_m captures the linear coefficients of the bounding line segments and has the following structure:

$$A_m = \begin{bmatrix} a_{11} & 0 & \cdots & \cdots & 0 \\ a_{21} & 0 & \cdots & \cdots & 0 \\ \vdots & 0 & \cdots & \cdots & 0 \\ a_{S_11} & 0 & \cdots & \cdots & 0 \\ 0 & a_{12} & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & \vdots & 0 & \cdots & 0 \\ 0 & a_{S_22} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & a_{1M} \\ 0 & \cdots & \cdots & 0 & \vdots \\ 0 & \cdots & \cdots & 0 & a_{S_MM} \end{bmatrix} \tag{9.3}$$

I_m is a sparse matrix with the same fill pattern as A_m where the a_{ij} entries are replaced with 1s.

G represents the gradient constraints

$$G = \begin{bmatrix} 1 & -1 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{9.4}$$

L represents the Laplacian constraints

$$L = \begin{bmatrix} -1 & 2 & -1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & -1 & 2 & -1 \end{bmatrix} \tag{9.5}$$

As shown in Chapter 4 the barrier method can be used to solve problems of this form. Again we proceed by minimizing $\phi(x,t) = tw^Tx - \sum_{i=1}^{m} \log(-(Ax-b)_i)$ for increasing values of t until convergence. At each value of t , the Newton step, needs to be computed. This involves the solution of a system of linear equations involving

the Hessian and the gradient of $\phi(x,t)$. The Hessian can be computed from the following expression $H = [A^T \text{diag}(s^{-2})A]$ where $s = b - Ax$ and s^{-2} denotes the vector formed by inverting and squaring the elements of s. Similarly the gradient of the $\phi(x,t)$ can be computed from the following expression :

$$g = -tw - A^T s^{-1} \tag{9.6}$$

Then the Newton step is computed by solving

$$[A^T \text{diag}(s^{-2})A]\delta x = g \tag{9.7}$$

Where $\delta x = [\delta d\, \delta y_m\, \delta y_g\, \delta y_l]^T$. The vector g can be split into four sections corresponding to the d, y_m, y_g and y_l variables as follows: $g = [\ g_d \quad g_m \quad g_g \quad g_l\ ]^T$ Similarly, divide the diagonal matrix, $\text{diag}(s^{-2})$ into 9 blocks, $D_1, \ldots D_9$, corresponding to the block structure of the matrix A.

It can be shown that the Hessian matrix $H = A^T \text{diag}(s^{-2})A$ takes on the following form: $=$

$$\begin{bmatrix} H_d & H_m^T & H_g^T & H_l^T \\ H_m & D_m & 0 & 0 \\ H_g & 0 & D_g & 0 \\ H_l & 0 & 0 & D_l \end{bmatrix} \tag{9.8}$$

where

$$\begin{aligned} H_d &= A_m^T D_1 A_m + B_d^T D_2 B_d + G^T D_g^+ G + L^T D_l^+ L \\ H_m &= -A_m^T D_1 I_m \\ H_g &= G D_g^- \\ H_l &= L D_l^- \end{aligned} \tag{9.9}$$

The diagonal matrices $D_g^+, D_g^-, D_l^+, D_l^-, D_g, D_l, D_m$ are given by the following expressions.

$$
\begin{aligned}
D_g^+ &= D_4 + D_5 && (9.10)\\
D_l^+ &= D_7 + D_8 && (9.11)\\
D_g^- &= D_5 - D_4 && (9.12)\\
D_l^- &= D_8 - D_7 && (9.13)\\
D_g &= D_g^+ + B_g^T D_6 B_g && (9.14)\\
D_l &= D_l^+ + B_l^T D_9 B_l && (9.15)\\
D_m &= -I_m^T D_1 I_m + B_m^T D_3 B_m && (9.16)
\end{aligned}
$$

At this point we observe that since the D_m, D_g and D_l matrices are all diagonal we can readily solve for δy_m, δy_g and δy_l in terms of δd by computing the Schur complement as follows: $\delta y_m = D_m^{-1}(g_m - H_m \delta d)$, $\delta y_g = D_g^{-1}(g_g - H_g \delta d)$, $\delta y_l = D_l^{-1}(g_l - H_l \delta d)$. Substituting these expressions back into the system yields the following expression where all of the auxiliary variables have been elided.

$$
\begin{aligned}
&(H_d - H_m^T D_m^{-1} H_m - H_g^T D_g^{-1} H_g - H_l^T D_l^{-1} H_l)\delta d \\
&\quad = (g_d - H_m^T D_m^{-1} g_m - H_g^T D_g^{-1} g_g - H_l^T D_l^{-1} g_l) \qquad (9.17)
\end{aligned}
$$

This can be written more concisely as follows:

$$
H'_d \delta d = g'_d \qquad (9.18)
$$

In short, computing the Newton Step boils down to solving the linear system in Equation 9.18. Note that the size of this system depends only on the dimension of the disparity vector, d. All of the auxiliary variables that were introduced have been removed which means that the computational complexity of this key stage is independent of the number of constraints that were used to construct the convex approximation.

The complete algorithm for solving the optimization problem is outlined in Algorithm 11.

Algorithm 11 Calculate the optimal disparity(d) for the problem in 9.1

choose an initial set of disparities and y_m, y_g and y_l
choose an initial t value and a μ value and set newton decrement $= \infty$
while $t \leq t_{\max}$ value **do**
  while Newton decrement is greater than a threshold **do**
    Compute the Hessian, H, from equation 9.8
    Compute the gradient, g, from equation 9.6
    Compute δd from equation 9.18
    Use δd to obtain δy_m,δy_g and δy_l.
    Compute Newton decrement $= g^T \delta x$
    Compute the step size β using line search
    Update d, y_m, y_g, y_l by $\beta\delta d$, $\beta\delta y_m$, $\beta\delta y_g$ and $\beta\delta y_l$ respectively.
  end while
  $t = \mu * t$
end while

9.1.1 Preconditioning strategies

Let us examine the structure of H'_d. H'_d contains scaled versions of G^TG, L^TL and $A_m^TA_m$. $A_m^TA_m$ is diagonal, G^TG is tridiagonal and L^TL is pentadiagonal. This means that L^TL is the dominant contributor to the sparsity structure of the matrix in question. Therefore the main step of the algorithm is simply the solution of a symmetric, positive definite pentadiagonal system.

As noted previously, the actual implementation of the algorithm involves smoothness terms in both the horizontal and vertical directions. This results in block pentadiagonal systems which have pentadiagonal blocks along the diagonal and diagonal blocks on the off diagonal. Essentially, the H'_d matrix associated with the two dimensional version of this formulation has the following structure:

$$\begin{bmatrix} A_1 & B_1^T & C_1^T & 0 & 0 \\ B_1 & \ddots & \ddots & \ddots & 0 \\ C_1 & \ddots & \ddots & \ddots & C_{n-2}^T \\ 0 & \ddots & \ddots & \ddots & B_{n-1}^T \\ 0 & 0 & C_{n-2} & B_{n-1} & A_n \end{bmatrix} \tag{9.19}$$

where the A_i are pentadiagonal and the B_i and the C_i are diagonal.

As we have seen before in Chapter 5, such structured matrices are ideal candidates for iterative solvers like conjugate gradients [33]. Also by performing incomplete Cholesky decomposition of these matrices in a manner similar to [19], good preconditioning can be achieved which significantly reduces the number of conjugate gradient iterations needed for convergence.

Let us try and first compute an exact Cholesky decomposition of a block pentadiagonal matrix. The factors will have to be lower block tridiagonal and upper block tridiagonal.

$$\begin{bmatrix} G_1 & 0 & 0 & 0 & 0 \\ F_1 & G_2 & 0 & 0 & 0 \\ E_1 & F_2 & G_3 & 0 & 0 \\ 0 & E_2 & F_3 & G_4 & 0 \\ 0 & 0 & E_3 & F_4 & G_5 \end{bmatrix} \begin{bmatrix} G_1^T & F_1^T & E_1^T & 0 & 0 \\ 0 & G_2^T & F_2^T & E_2^T & 0 \\ 0 & 0 & G_3^T & F_3^T & E_3^T \\ 0 & 0 & 0 & G_4^T & F_4^T \\ 0 & 0 & 0 & 0 & G_5^T \end{bmatrix} = \begin{bmatrix} A_1 & B_1^T & C_1^T & 0 & 0 \\ B_1 & A_2 & B_2^T & C_2^T & 0 \\ C_1 & B_2 & A_3 & B_3^T & C_3^T \\ 0 & C_2 & B_3 & A_4 & B_4^T \\ 0 & 0 & C_3 & B_4 & A_5 \end{bmatrix} \tag{9.20}$$

Similar to the Concus and Golub preconditioner, the following set of equations are obtained.

$$G_1 G_1^T \equiv Q_1 = A_1 \tag{9.21}$$

$$P_1 \equiv B_1 \tag{9.22}$$

$$F_1 = P_1 G_1^{-T} \tag{9.23}$$

$$G_2 G_2^T \equiv Q_2 = A_2 - F_1 F_1^T \tag{9.24}$$

$$E_1 = C_1 G_1^{-T} \tag{9.25}$$

$$P_2 = B_2 - E_1 F_1^T \tag{9.26}$$

$$F_2 = P_2 G_2^{-T} \tag{9.27}$$

$$G_3 G_3^T \equiv Q_3 = A_3 - F_2 F_2^T - E_1 E_1^T \tag{9.28}$$

The problem as in the case of the Cholesky decomposition of block tridiagonal matrices seen in section 5.2.4 is that while Q_1 is guaranteed to be pentadiagonal and hence easily factorizable, the Q_i, $i \geq 2$ are not guaranteed to be structured in any manner. Hence an approximation needs to be made so that each Q_i has definite structure in terms of the number of non-zero diagonals. This is accomplished by simply limiting the number of non-zero diagonals of Q_i that are retained at every step. The larger the number of diagonals chosen, the higher the complexity of the Cholesky factorization into G_i, but the better the pre-conditioning effect. The trade-off, of course, is that more non-zero diagonals necessitates more storage. In practice storing 10 non-zero diagonals seems to provide an appropriate tradeoff between the effectiveness of the preconditioner and the storage footprint of the algorithm.

9.2 Stereo implementation

The first step in our stereo matching procedure is to compute a correlation volume which encodes for each pixel the costs associated with all legal disparities. For this purpose the Adaptive Matching Score method described by Yoon and Kweon in [98] is employed. In this implementation 33x33 windows were used along with the

following parameter values: $\gamma_p = 36$, $\gamma_c = 7$ and $T = 40$. Prior to computing the support weights in each window, the color image was smoothed using a 5x5 median filter to enhance color homogeneity while respecting image edges.

Once the match scores had been computed for all integer disparities, a quadratic fit was used in the vicinity of local minima to establish subpixel offset and subpixel score values. This resulted in a set of (disparity, score) values for each pixel where the disparity values are not all integral. The convex approximation procedure described in section 8.3.1 was then applied to the score function associated with every pixel to produce a piecewise linear convex lower approximation for use in the optimization procedure.

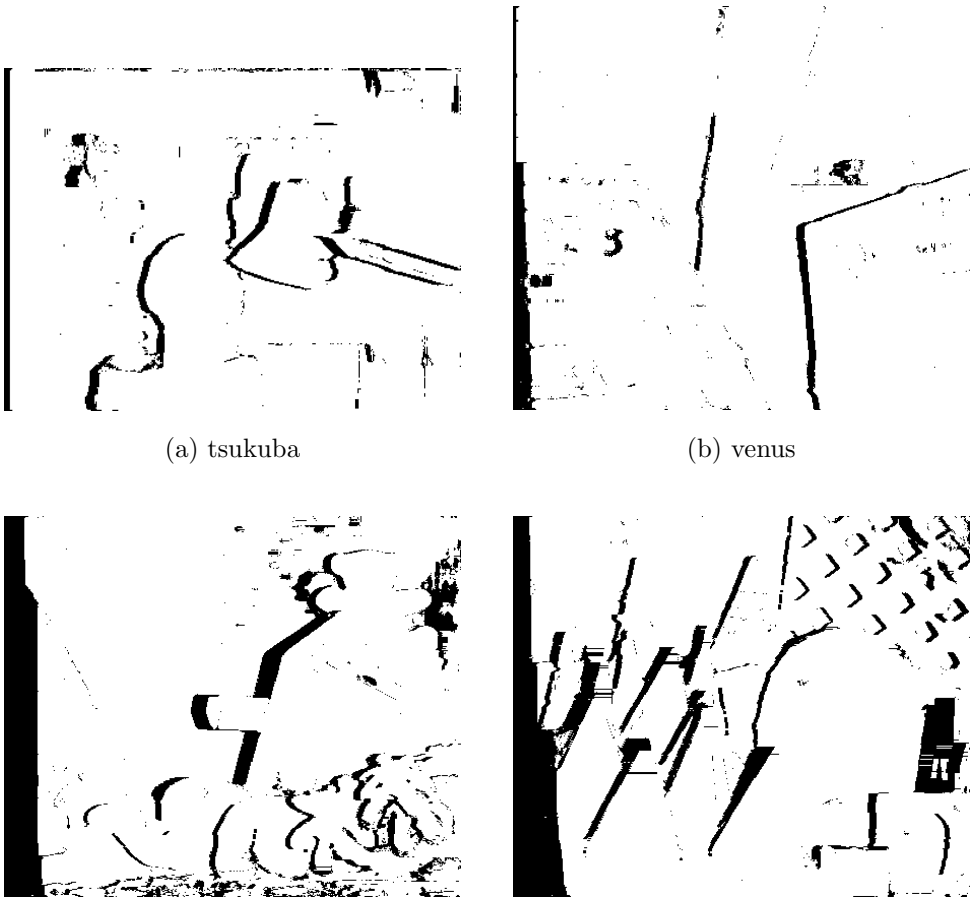
The structure of the constraint system associated with the LP is determined by these convex approximations and the gradient and Laplacian equations which are fixed. What remains then is to specify the weights associated with the match terms, w_m , the gradient terms, w_g and the Laplacian terms, w_l.

The match weights associated with each pixel should represent the degree of confidence in that pixels match score. The reasoning here is that pixels that are occluded or otherwise suspicious should have low w_m values reflecting the fact that their match costs should be devalued while the weights associated with pixels whose disparities are more certain should be enhanced.

We begin by using the initial disparity estimates provided by the Yoon and Kweon matcher to decide on which pixels appear reliable by using a simple left right check. Additionally, we use the disparity discontinuities found in the right image to predict occlusions in the left image and vice versa in a manner similar to the approach described in Sun *et al* . [87]. The resulting match weights w_m essentially represent an approximation for the occluded regions in the image.

Figure 9.1 shows what these match weight images look like for various images.

The weights associated with the gradient and Laplacian term are computed by considering the color differences between neighboring pixels. Similar weighting terms



(a) tsukuba (b) venus

(c) teddy (d) cones

Figure 9.1: Match weights computed for the four images of the Middlebury dataset. These weights represent an estimate for the occlusion regions in the image.

are used in most modern stereo matching algorithms. In this particular case the weight associated with the gradient term, y_{g_i}, is computed as follows

$$w_{gi} = s_g \exp(-(\Delta c_i/\sigma)^2) \tag{9.29}$$

Where Δc_i represents the Euclidean difference between the colors associated with pixel i and pixel $i+1$ in Lab space. The weight associated with the Laplacian term y_{li} is computed as follows:

$$w_{li} = s_l \exp(-((\Delta c_{i-1}/\sigma)^2 + (\Delta c_i/\sigma)^2)/2) \tag{9.30}$$

Note that the Laplacian weight considers the difference between pixel i and $i-1$ as well as the difference between pixels i and $i+1$. In this implementation the parameter values were set as follows: $s_g = 1, s_l = 2, \sigma = 6$. Once again the image is smoothed with a 5x5 median filter before the color differences, Δc_i, are computed.

Chapter 10

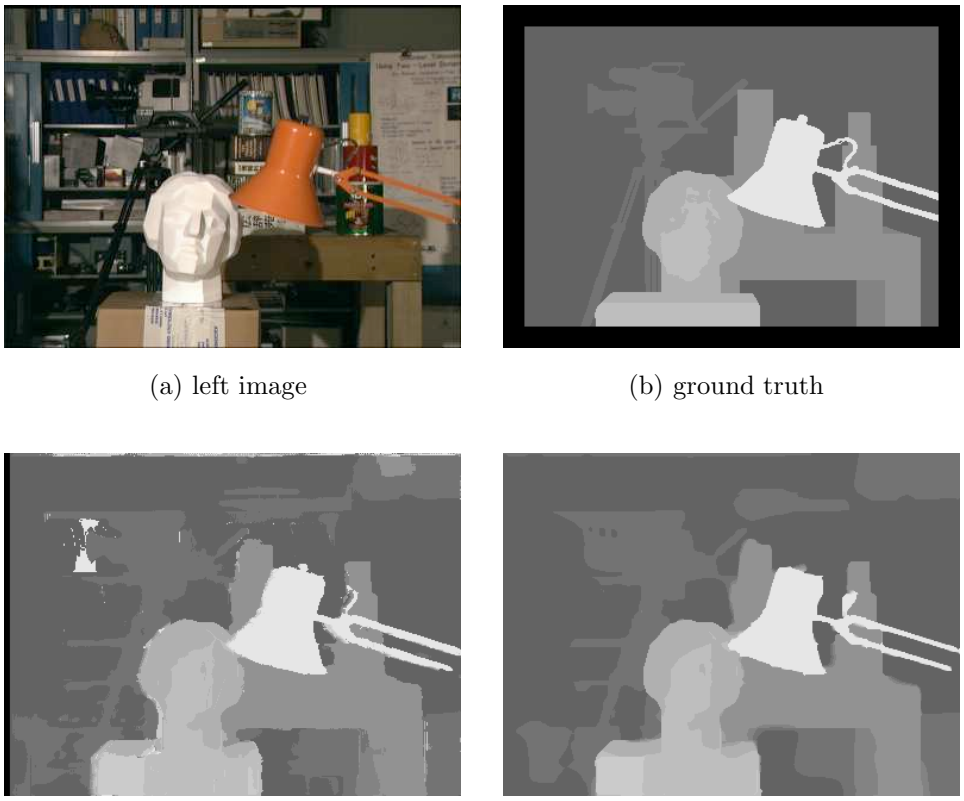
Stereo results

The proposed method was applied to the Middlebury data set [81] and the results are summarized in figures 10.1, 10.2, 10.4 and 10.3. Each of the figures show the left image of the stereo pair, the ground truth disparity, the result of our implementation of the Adaptive Support weight method and the result obtained after the proposed optimization scheme has been applied.

Although the theory developed incorporates both penalties on the gradient and the Laplacian, on the current Middlebury evaluation the performance with or without the Laplacian term seems to be the same. This seems to indicate incorrent weighting more that it seems to be a case of the unnecessary term being added. Further analysis needs to be done to weight the gradient and Laplacian penalties appropriately.

Pixels that have errors exceeding a 0.5 pixel threshold are shown in figure 10.5. Note that this is the strictest possible threshold in the Middlebury dataset.

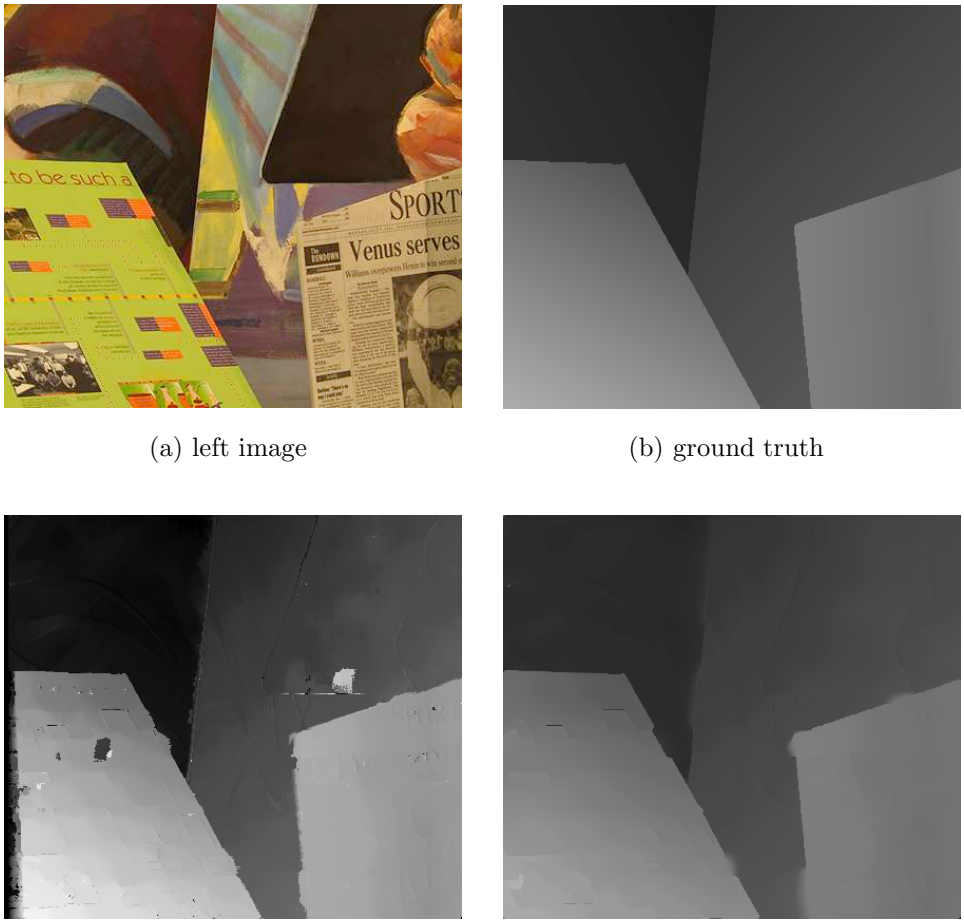
The entire procedure was implemented in Matlab and the time required to perform the LP optimization was 8 minutes on a Intel Core 2Duo system with 2GB of memory. During this procedure the system performed 40 Newton iterations where each step took approximately 13.4 seconds. Again the vast majority of the computational effort is spent inverting the sparse system in equation 9.18. As is typical of the barrier method, the system converges to an answer quite rapidly.



(a) left image (b) ground truth

(c) adaptive score (d) our result

Figure 10.1: Result on the Tsukuba image. The figure shows the left image, the ground truth, the result obtained using the adaptive score method and the final result of the proposed method.



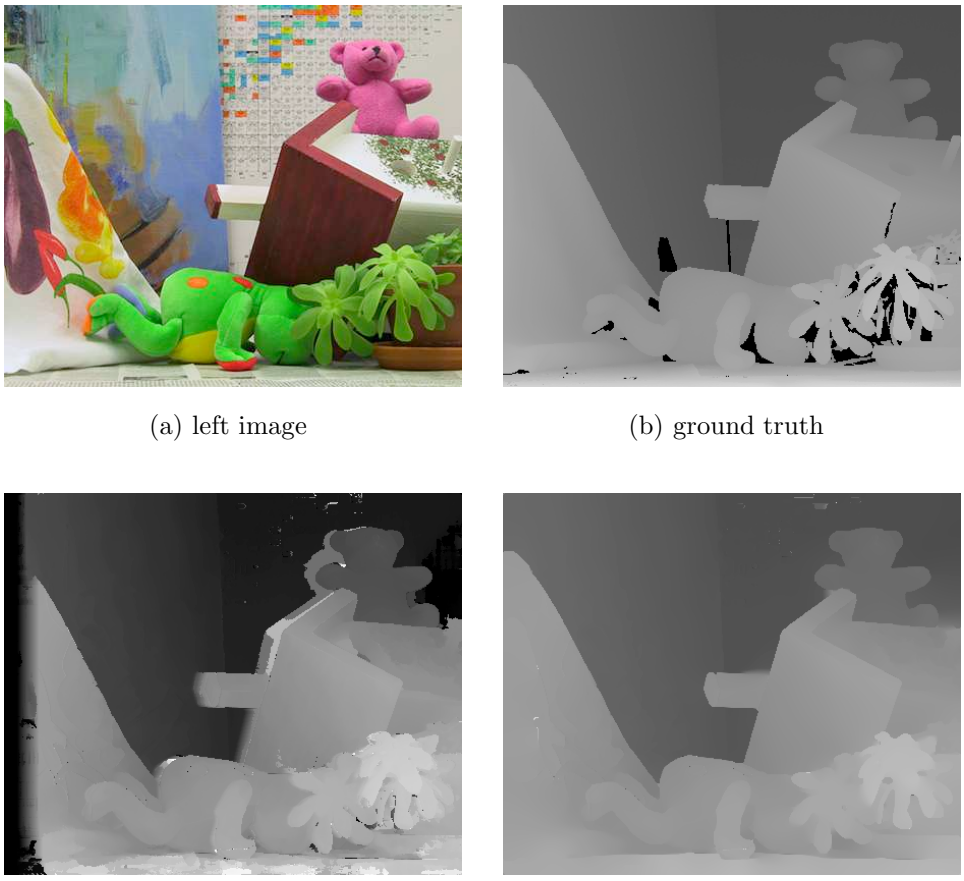
(a) left image

(b) ground truth

(c) adaptive score

(d) our result

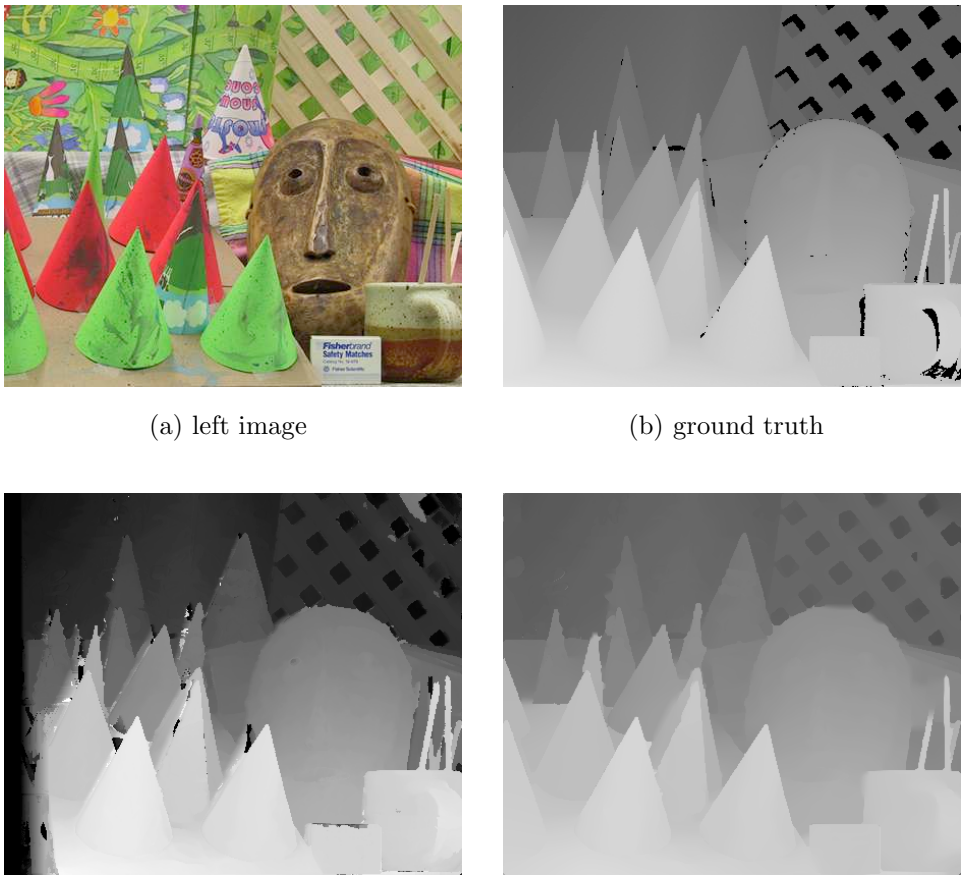
Figure 10.2: Result on the Venus image. The figure shows the left image, the ground truth, the result obtained using the adaptive score method and the final result of the proposed method.



(a) left image (b) ground truth

(c) adaptive score (d) ground truth

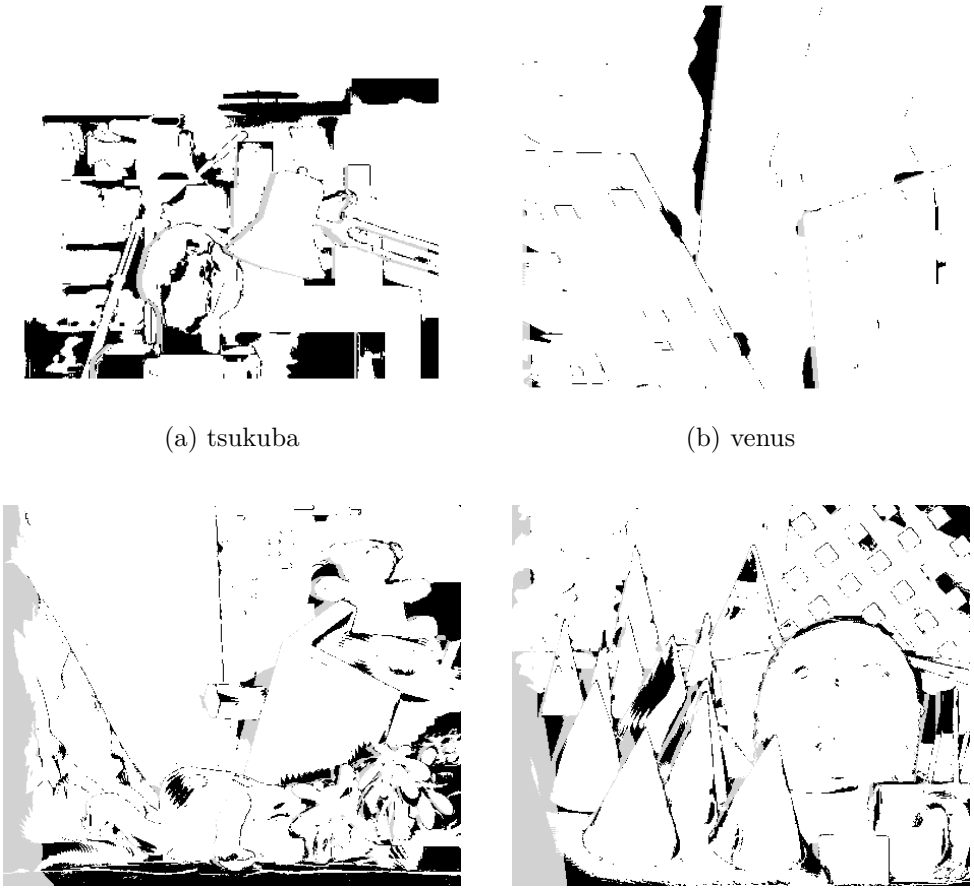
Figure 10.3: Result on the Teddy image. The figure shows the left image, the ground truth, the result obtained using the adaptive score method and the final result of the proposed method.



(a) left image (b) ground truth

(c) adaptive score (d) our result

Figure 10.4: Result on the Cones image. The figure shows the left image, the ground truth, the result obtained using the adaptive score method and the final result of the proposed method.



(a) tsukuba (b) venus

(c) teddy (d) cones

Figure 10.5: Evaluation of the results obtained by the interior point algorithm - pixels having a disparity greater than 0.5 away from ground truth are indicated.

Table 10.1: Ranking of the new algorithm in the Middlebury evaluation table with a 0.5 pixel error threshold for the Tsukuba and Venus images.

	Tsukuba			Venus		
Methods	nonocc	all	disc	nonocc	all	disc
SubPixDoubleBP [96]	8.78	9.45	14.9	0.72	1.12	5.24
C-SemiGlobal [44]	13.9	14.7	18.9	3.3	3.82	10.9
ImproveSubPix [31]	8.96	9.66	16.2	4.62	5.41	16.9
OverSegBP [101]	7.75	8.17	13.8	4.33	4.73	16.8
SemiGlob [43]	13.4	14.3	20.3	4.55	5.38	15.7
GenModel [86]	7.89	10.0	18.5	4.59	6.03	23.5
LP	**18.7**	**19.3**	**20.8**	**3.0**	**3.71**	**18.8**
LP with no lap	**19.2**	**19.6**	**18.1**	**2.75**	**3.65**	**16.7**
Adaptive Weight (our)	19.8	21.4	18.6	3.67	5.28	16.9

Table 10.1 shows the percentage of erroneous pixels in the final result with a 0.5 pixel threshold and compares it against other state of the art methods for the Tsukuba and Venus images. It also shows shows how the proposed optimization method improves upon a naive interpretation of the match scores returned by the Yoon and Kweon matcher. Table 10.2 does the same for the Cones and Teddy images.

10.1 Potential future work

There are a number of directions that could be taken to further improve the results on the given data sets. The current system performs a single pass to arrive at the final result. One can imagine a system that proceeds in several rounds where the results of one round are used to refine the weights used in the next round.

Other methods make use of explicit color segmentations and plane fitting methods. These approaches are particularly effective on this data set where planar surfaces predominate. This type of analysis could easily be incorporated into the overall system. Explicit segmentations could also serve to improve performance on and

Table 10.2: Ranking in the Middlebury evaluation table with a 0.5 pixel error threshold for the Teddy and Cones images

	Teddy			Cones		
Methods	nonocc	all	disc	nonocc	all	disc
SubPixDoubleBP [96]	10.1	16.4	21.3	8.49	14.7	16.5
C-SemiGlobal [44]	9.82	17.4	22.8	5.37	11.7	12.8
ImproveSubPix	11.0	17.8	24.1	4.90	10.8	12.2
OverSegBP [101]	13.2	19.3	27.5	6.53	12.6	14.0
SemiGlob [43]	11.0	18.5	26.1	4.93	12.5	13.5
GenModel [86]	14.8	22.8	31.8	10.2	20.2	19.0
LP	**13.5**	**19.2**	**29.7**	**9.57**	**16.2**	**20.3**
LP with no lap	**13.1**	**18.3**	**28.2**	**9.60**	**17.2**	**19.1**
Adaptive Weight (our)	14.0	22.9	28.0	9.24	19.5	18.3

around occlusion boundaries.

While the Matlab implementation does a reasonable job of performing a large scale optimization on a quarter of a million continuous variables, we feel that there is ample room for further improvement in this area. More specifically the structure of equation 9.18 makes it particularly amenable to parallel solution methodologies that could exploit the abundance of floating point performance afforded by GPUs and other parallel architectures in a manner very similar to the graph cuts implementation.

Chapter 11

Conclusion

This thesis has described mechanisms for tackling some energy minimization problems in computer vision by converting them into linear programs. The theory of convex optimization was then used to find efficient solutions to the linear programs. Vision problems have a lot of inherent structure to them and this can be exploited to provide efficient algorithms.

A general methodology was put forth in this thesis which converts energy minimization problems into linear programs and then uses the interior point barrier method to solve them. As a result of solving for the Newton steps in the barrier method, the bottleneck step becomes the solution of large number of linear equations. The method is most applicable for the cases where these linear equations have sparseness structure associated with them.

Two different problems were tackled using this methodology. The first problem was the discrete graph cuts problem. Specifically we used the method to convert this discrete problem to the continuous problem of unconstrained ℓ_1 norm minimization. This was then used for the interactive foreground-background segmentation. The procedure developed was highly parallelizable and the thesis described an implementation on a GPU.

Importantly, although the formulation of the graph is slightly different from the

Table 11.1: Single Precision Floating Point performance afforded by a range of current CPUs and GPUs

Processor	**GFLOPS**
Intel Core 2 Duo	8
NVIDIA 7800	35
ATLI R580	125
NVIDIA 8800	330
Sony/IBM Cell	205

one used in Kolmogorov and Zabih [58], it has the ability to deal with the exact same set of energy functions. Therefore any energy minimization problem that is solved via α expansion and hence via repeated computation of a min-cut can be solved using the ℓ_1 norm minimization.

The main advantage of viewing graph cuts from this continuous perspective is that it provides a more direct understanding of the intermediate stages in the creation of a mincut. It also manages to connect the problem to other areas of active research. ℓ_1 norm minimization has been studied in other areas such as ℓ_1 regularized logistic regression [54], robust subspace computation [51], motion analysis [5] etc. The linear equations being solved are very similar to Poisson equations which have been extensively studied in fields such as mechanical engineering.

Considering the remarkable advances in CPU and GPU performance that have been enabled by decreasing feature size and increasing parallelism, the new graph cuts algorithm should be able to exploit the upcoming hardware products. 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. Current GFLOPS values of some of the popular parallel architectures are shown in table 11.1. The proposed implementation scheme is designed to exploit the performance afforded by these types of systems.

The other problem that was tackled was stereo matching. The cost function for

stereo was approximated by a convex lower bound and once again a linear program was formulated. Solution via the barrier method led to sparse and structured linear equation systems.

There are two main observations underlying this approach. The first is that the proposed convex functions provide a reasonable model of the true objective function in most cases and effectively captures the ambiguities inherent in the data. The second is that the Hessian matrices associated with the resulting optimization problems have special structure which mirrors the clique structure of the underlying objective function.

Stereo matching involves variables that are inherently continuous and the usage of convex optimization provides one with the ability to directly incorporate this aspect of the problem. While this thesis concentrates on the application of the convex approximation to stereo, one can extend the idea to other problems that have similar matching goals. Therefore problems such as motion, parametric warping etc could also be handled.

The linear programs that are formed during the course of solving both the graph cuts problem and stereo matching contain an extremely large number of variables and constraints. They are beyond the reach of state of the art LP solvers since they are unable to identify the connectivity structure present in the MRFs. To solve them, one has to be able to exploit this structure. Therefore, a major contribution of this thesis is the ability of the methods to transfer the connectivity structure of an MRF into the bandedness of a sparse matrix. In essence, the crucial step of the method always is the solution of sparse and structured linear equations.

In conclusion, the thesis provides a very effective technique for structured energy minimization problems. The structure is exploited to efficiently solve the normal equations that arise out of interior point barrier methods. This leads to algorithms that are amenable to parallelism. In certain cases, the continuous representation models the problem in a better manner. Overall, the area of energy minimization

for large scale MRFs has been dominated by discrete optimization techniques. Using the methods in this thesis, one has a plausible alternative that uses continuous optimization.

Bibliography

[1] `http://www.mosek.com`.

[2] `http://tomopt.com/tomlab`.

[3] C. Akcadogan and H. Dag. A parallel implementation of chebyshev preconditioned conjugate gradient method. In *Second International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 1–8, 2003.

[4] K. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[5] M. Ben-Ezra, S. Peleg, and M. Werman. Real-time motion analysis with linear programming. *Computer Vision and Image Understanding*, 78(1):32–52, 2000.

[6] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:401–406, 1998.

[7] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr. Interactive image segmentation using an adaptive gmmrf model. In *European Conference on Computer Vision*, pages 428–441, 2004.

[8] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. In *Special Interest Group on Graphics and Interactive Techniques(SIGGRAPH)*, pages 917–924, 2003.

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

[10] Y. Boykov and M.P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *IEEE International Conference of Computer Vision*, pages 105–112, 2001.

[11] 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*, 26:1124–1137, Sept 2004.

[12] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, Nov 2001.

[13] P. Bremaud. *Markov Chains Gibbs Fields, Monte Carlo Simulation and Queues*. Springer-Verlag, 1999.

[14] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision*, pages 25–36, 2004.

[15] A. Brunton, C. Shu, and G. Roth. Belief propagation on the gpu for stereo vision. In *Third Canadian Conference on Computer and Robot Vision*, 2006.

[16] D. Burschka, S. Lee, and G. Hager. Stereo-based obstacle avoidance in indoor environments with active sensor re-calibration. In *International Conference on Robotics and Automation*, pages 2066–2072, 2002.

[17] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Symposium on Discrete Algorithms*, pages 109–118, 2001.

[18] F. Chen, K. Theobald, and G. Gao. Implementing parallel conjugate gradient on the earth multithreaded architecture. In *IEEE International Conference on Cluster Computing*, pages 459–469, 2004.

[19] P. Concus, G. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM Journal of Scientific Computing*, 6(1):220–252, 1985.

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

[21] T. Cour and J. Shi. Solving markov random fields with spectral relaxations. In *International Workshop on Artificial Intelligence and Statistics*, 2007.

[22] A. Criminisi, G. Cross, A. Blake, and V. Kolmogorov. Bilayer segmentation of live video. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 53–60, 2006.

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

[24] E.Boros and P.Hammer. Pseudo boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225, 2002.

[25] S. Farlow. *Partial Differential Equations for Scientists and Engineers*. John Wiley and Sons, 1982.

[26] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 261–268, 2004.

[27] R. Fernando and M. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

[28] J. Fung, S. Mann, and C.Aimone. Openvidia: parallel gpu computer vision. In *ACM Multimedia*, pages 849–852, 2005.

[29] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000.

[30] W. Gander and G. Golub. Cyclic reduction - history and applications. In *Workshop on Scientific Computing*, 1997.

[31] S. Gehrig and U. Franke. Improving sub-pixel accuracy for long range stereo. In *ICCV Virtual Representations and Modeling of Large-Scale Environments workshop*, 2007.

[32] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:452–472, 1984.

[33] G. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.

[34] M. Gong and Y. Yang. Near real-time reliable stereo matching using programmable graphics hardware. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 924–931, 2005.

[35] N. Goodnight, C. Woolley, G. Lewin, D. Luebke, and G. Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *SIGGRAPH conference on Graphics Hardware*, pages 102–111, 2003.

[36] L. Grady and G. Funka-Lea. Multi-label image segmentation for medial applications based on graph-theoretic electrical potentials. In *ECCV 2004 Workshops CVAMIA and MMBIA*, pages 230–245, 2004.

[37] L. Grady, T. Schiwetz, S. Aharon, and R. Westermann. Random walks for interactive organ segmentation in two and three dimensions: Implementation

and validation. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, pages 773–780, 2005.

[38] K. Gray. *Microsoft DirectX9 Programmable Graphics Pipeline*. Microsoft, 2003.

[39] A. Griesser, S.D. Roeck, A. Neubeck, and L.V. Gool. Gpu-based foreground-background segmentation using an extended colinearity criterion. In *Vision Modeling and Visualization*, pages 319–326, 2005.

[40] J.M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. unpublished, 1971.

[41] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[42] M. Heroux, P. Vu, and C. Yang. A parallel preconditioned conjugate gradient package for solving sparse linear systems on a cray y-mp. *Applied Numerical Mathematics*, 8(2):93–115, 1991.

[43] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 807–814, 2005.

[44] H. Hirschmuller. Stereo vision in structured environments by consistent semi-global matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2386–2393, 2006.

[45] L. Hong and G. Chen. Segment-based stereo matching using graph cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 74–81, 2004.

[46] H. Ishikawa. Exact optimization for markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:1333–1336, 2003.

[47] H. Ishikawa and D. Geiger. Occlusions, discontinuities and epipolar lines in stereo. In *European Conference on Computer Vision*, pages 232–248, 1998.

[48] H. Jiang, M. Drew, and Z.N. Li. Matching by linear programming and successive convexification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):959–975, 2007.

[49] O. Juan and Y. Boykov. Active graph cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1023–1029, 2006.

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

[51] Q. Ke and T. Kanade. Robust subspace computation using l1 norm. Technical report, Carnegie Mellon University, 2003.

[52] A Klaus, M Sormann, and K Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *International Conference on Pattern Recognition*, pages 15–18, 2006.

[53] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *Journal of the ACM*, 49(5):616–639, 2002.

[54] K. Koh, S. Kim, and S. Boyd. An interior-point method for large-scale l1 regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, 2006.

[55] P. Kohli and P. Torr. Efficiently solving dynamic markov random fields using graph cuts. In *IEEE International Conference of Computer Vision*, pages 922–929, 2005.

[56] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006.

[57] V. Kolmogorov and R. Zabih. Computing visual correspondance with occlusions via graph cuts. In *IEEE International Conference of Computer Vision*, pages 508–515, 2001.

[58] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):1124–1137, 2004.

[59] N. Komodakis and G. Tziritas. A new framework for approximate labeling via graph cuts. In *IEEE International Conference of Computer Vision*, pages 1018–1025, 2005.

[60] N. Komodakis, G. Tziritas, and N. Paragios. Fast approximately optimal solutions for single and dynamic mrfs. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[61] J. Krüger and R. Westermann. Linear algebra operators for gpu implementation of numerical algorithms. In *Special Interest Group on Graphics and Interactive Techniques(SIGGRAPH)*, 2003.

[62] A. Kumar, S. Haker, C. Vogel, A. Tannenbaum, and S. Zucker. Stereo disparity and l1 minimization. In *Conference on Decision and Control*, pages 1125–1129, 1997.

[63] M.P. Kumar, V. Kolmogorov, and P.H.S. Torr. An analysis of convex relaxations for map estimation. In *Neural Information Processing Systems*, 2007.

[64] M.P. Kumar, P.H.S. Torr, and A. Zissermann. Solving markov random fields using second order cone programming relaxations. In *IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition, pages 1045–1052, 2006.

[65] X. Lan, S. Roth, D.P. Huttenlocher, and M.J. Black. Efficient belief propagation with learned higher-order markov random fields. In *European Conference on Computer Vision*, pages 269–282, 2006.

[66] L.Ford and D.Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[67] S.Z. Li. *Markov Random Field Modeling in Image Analysis*. Springer-Verlag, 2001.

[68] W. Li and J. Swetits. The linear ℓ_1 estimator and the huber m-estimator. *SIAM Journal of Optimization*, 8(2):457–475, 1998.

[69] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *IEEE International Conference of Computer Vision*, pages 259–265, 2005.

[70] S. Madarasmi, D. Kersten, and T. Pong. Multi layer surface segmentation using energy minimization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 774–775, 1993.

[71] O. Mangasarian and D. Musicant. Robust linear and support vector regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):950–955, 2000.

[72] R. Merris. Laplacian matrices on graphs: A survey. *Linear Algebra and its Applications*, 197/198:143–176, 1994.

[73] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A.E. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

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

[75] M.C. Pinar. Finite computation of the ℓ_1 estimator from huber's m-estimator in linear regression. *Computing*, 72:365–384, 2004.

[76] P. RaviKumar and J. Lafferty. Quadratic programming relaxations for metric labeling and markov random field map estimation. In *International Conference of Machine Learning*, pages 737–744, 2006.

[77] R.J. Rost. *OpenGL Shading Language*. Addison-Wesley, 2006.

[78] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. In *Special Interest Group on Graphics and Interactive Techniques(SIGGRAPH)*, pages 309–314, 2004.

[79] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *IEEE International Conference of Computer Vision*, pages 492–502, 1998.

[80] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2002.

[81] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 195–202, 2003.

[82] J. Shewchuk. An introduction to conjugate gradient method without the agonizing pain. Technical Report CS-94-125, Carnegie Mellon University, 1994.

[83] S. Sinha and M. Pollefeys. Multi-view reconstruction using photo-consistency and exact silhoutte constraints: A maximum flow formulation. In *IEEE International Conference of Computer Vision*, pages 349–356, 2005.

[84] A.K. Sinop and L. Grady. A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *IEEE International Conference of Computer Vision*, 2007.

[85] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 345–352, 2000.

[86] C Strecha, R Fransens, and L.V. Gool. Combined depth and outlier estimation in multi-view stereo. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2934–2401, 2006.

[87] J. Sun, Y. Li, and S.B. Kang. Symmetric stereo matching for occlusion handling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 399–406, 2005.

[88] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparitive study of energy minimization methods for markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[89] M.F. Tappen and W.T. Freeman. Comparison of graph cuts with belief propagation for stereo using identical mrf parameters. In *IEEE International Conference of Computer Vision*, pages 900–907, 2003.

[90] P.H.S. Torr. Solving markov random fields using semi definite programming. In *International Workshop on Artificial Intelligence and Statistics*, 2003.

[91] M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky. Tree reweighted belief propagation and approximate ml estimation by pseudo-moment matching. In *International Workshop on Artificial Intelligence and Statistics*, 2003.

[92] M.J. Wainwright, T.S. Jaakkola, and A.S. Willsky. Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. *Statistics and Computing*, 14(2):143–166, 2004.

[93] J. Wang and E. Adelson. Layered representation for motion analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 361–366, 1993.

[94] Y. Weiss and W. Freeman. On the optimality of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):723–735, 2001.

[95] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister. Stereo matching with color-weighted correlation, hierarchical belief propagation and occlusion handling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2347 – 2354, New York, 2006.

[96] Q Yang, R Yang, J Davis, and D Nister. Spatial-depth super resolution for range images. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[97] C. Yanover, T. Meltzer, and Y. Weiss. Linear programming relaxations and belief propagation - an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

[98] K.J. Yoon and I.S. Kweon. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):650–656, 2006.

[99] K.J. Yoon and I.S. Kweon. Stereo matching with the distinctive similarity measure. In *IEEE International Conference of Computer Vision*, 2007.

[100] F. Zhang. *The Schur Complement and its Applications*. Springer, 2005.

[101] C L Zitnick and S B Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision*, 75(1):49–65, 2007.