GRAPH BASED ALGORITHMS FOR SCENE RECONSTRUCTION FROM TWO OR MORE VIEWS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Vladimir Kolmogorov

January 2004

GRAPH BASED ALGORITHMS FOR SCENE RECONSTRUCTION FROM TWO OR MORE VIEWS

Vladimir Kolmogorov, Ph.D.

Cornell University 2004

In recent years, graph cuts have emerged as a powerful optimization technique for minimizing energy functions that arise in low-level vision problems. Graph cuts avoid the problems of local minima inherent in other approaches (such as gradient descent). The goal of this thesis is to apply graph cuts to a classical computer vision problem — scene reconstruction from multiple views, i.e. computing the 3-dimensional shape of the scene.

This thesis provides a technical result which greatly facilitates the derivation of the scene reconstruction algorithm. Our result should also be useful for developing other energy minimization algorithms based on graph cuts. Previously such algorithms explicitly constructed graphs where a minimum cut also minimizes the appropriate energy. It is natural to ask for what energy functions we can construct such a graph. We answer this question for the class of functions of binary variables that can be written as a sum of terms containing three or fewer variables. We give a simple criterion for functions in this class which is necessary and sufficient, as well as a necessary condition for any function of binary variables. We also give a general-purpose graph construction for functions that satisfy our criterion.

Using this result, we develop several algorithms for the scene reconstruction problem. We formulate this problem in an energy minimization framework. The

energy that we minimize encodes all the constraints of the problem: it treats the input images symmetrically, handles visibility properly, and imposes spatial smoothness while preserving discontinuities. We discuss several smoothness terms that we can use. Experimental results show that our algorithms for two views are among the top performers, and that the accuracy of the reconstruction is greatly improved when more views are used.

In the last chapter of the thesis we address the issue of the efficiency of min-cut/max-flow algorithms for computer vision applications. We compare the running times of several standard algorithms, as well as a new algorithm that we have recently developed. We benchmark these algorithms on a number of typical graphs arising in vision. In many cases our new algorithm works several times faster than other methods.

BIOGRAPHICAL SKETCH

Vladimir Kolmogorov was born in Tula, Russia, in 1977. In June 1999 he received the M.S. degree from the Moscow Institue of Physics and Technology in applied mathematics and physics. In the fall of 1999 he entered the Ph.D. program in the Field of Computer Science at Cornell University. In August 2002 he received the M.S. degree in computer science.

ACKNOWLEDGEMENTS

First of all I would like to express my gratitude to my supervisor Ramin Zabih with whom I have been fortunate to work during the last four years. Without his guidance, support, patience and encouragement this work could not have been done. Much of what I have learned about computer vision I owe to his vast knowledge of the field. His intuition and ability to identify modern directions in vision allowed us to win a prestigous award at the European Conference on Computer Vision.

The influence of Yuri Boykov on my growth as a researcher cannot be overestimated. His perspective on computer vision helped me to look at many problems from a different point of view. I would like to thank him for hours of discussions about computer vision and other random topics which produced many new ideas.

I am indebted to Olga Veksler who was always available to answer any question I had. Her clear undertanding of problems was always extremely helpful. She also proof-read and vastly improved many unreadable manuscripts.

I am grateful to my supervisor at Cornell Medical School Yi Wang for introducing me to the exciting field of magnetic resonance imaging, as well as for being an excellent travel companion in trips to China and Hawaii. His deep understanding of MRI physics proved to be invaluable in our joint work on reducing motion artifacts.

Two internships at Siemens Corporate Research gave me a valuable industrial experience and yielded new ideas which have become a part of this thesis. I would like to thank Yuri Boykov, Alok Gupta and Gareth Funka-Lea whose support made it possible to happen.

I am grateful to my collaborator Junhwan Kim. His ability to come up with new interesting problems and his meticulous approach were absolutely crucial for our joint work on stereo and mutual information.

I would like to thank Eva Tardos and Jon Kleinberg. Being the world's top experts in the field of combinatorial optimization, they were very influencial in shaping my approach to computer vision.

I would like to thank Steven Gortler whose insightful comments made it possible to unify two scene reconstruction algorithms presented in this thesis and make this work more complete.

I would like to thank all members of my committee (Ramin Zabih, Eva Tardos, Steven Chase, Yuri Boykov and Olga Veksler) who kindly agreed to be the official examiners of this thesis.

I would like to thank all people who contributed datasets for our experiments: Y. Ohta and Y. Nakamura from University of Tsukuba, Daniel Scharstein from Middlebury College, and Rick Szeliski from Microsoft Research. I would also like to thank Daniel and Rick for including our results in their evaluation of stereo algorithms.

My research has been supported by NSF grants IIS-9900115 and CCR-0113371, by a grant from Microsoft Research, and by internships at Siemens Corporate Research. Chapter 3 of this thesis is based on joint work with Ramin Zabih, chapter 4 on joint work with Ramin Zabih and Steven Gortler, and chapter 5 on joint work with Yuri Boykov.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Energy minimization in vision

Energy minimization is a natural framework for many vision applications. It has several advantages. It allows a clean specification of the problem to be solved, as distinct from the algorithm used to solve it. In addition, energy minimization naturally allows the use of soft constraints, such as spatial coherence. In an energy minimization framework, it is possible to cause ambiguities to be resolved in a manner that leads to a spatially smooth answer. Finally, energy minimization avoids being trapped by early hard decisions.

The energy minimization approach is very popular in vision, and has been widely applied to many problems. The amount of literature on the subject is immense, see for example [HS81] for optical flow; [GG84] for image restoration; [BZ87] for surface reconstruction; [Bar89] for stereo matching; [DE87] and [HPB98] for texture segmentation; [TP86] for edge detection. For a brief overview of energy minimization formulation for many vision problems see [PTK85].

Solving a problem via an energy minimization consists of two major steps. First an *objective* function is formulated. It maps all possible solutions to real numbers, and it shows how good (or bad) a candidate solution is. An objective function is usually a sum of terms corresponding to different constraints of the problem, either soft or hard. In this thesis, all objective functions will measure the badness of a candidate solution. We will refer to these objective functions as *energy* functions

The second step of the approach is to minimize the energy function. This is often a very hard task. Energy functions that arise in vision usually have thousands of dimensions and many local minima. Many researchers have tried to use some general minimization techniques such as gradient descent or simulated annealing. The former technique can be applied to almost any energy function of continuous variables, and the latter to almost any function of discrete variables. However, the price paid for this generality is that they often produce very poor results since they get stuck in local minima, or they take an extremely long time to converge. Moreover, because annealing is theoretically capable of finding the global minimum of an arbitrary energy function, it requires exponential time.

Recently, new fast energy minimization techniques based on graph cuts have emerged. These techniques can be applied to a restricted class of energy functions of discrete variables. An advantage of these methods is that in certain cases they can produce a *global* minimum of the energy, or in other cases a local minimum with some strong properties. A brief overview and history of such techniques in computer vision is given in the next section.

Although graph cuts are a powerful optimization technique, they can be applied only to certain types of constraints. Two common constraints that researchers have successfully incorporated into the graph cut framework are data and smoothness constraints. The former constraint prefers solutions that explain well the observed data, and the latter constraint prefers spatially smooth solutions.

In this thesis we develop a new graph cut approach to the problem of scene reconstruction from images taken from multiple views. This problem has its own additional constraints described later in the thesis. We show how to incorporate these constraints into a graph cut framework. As a result, we demonstrate that the resulting algorithms give significantly more accurate reconstructions.

Note that because graph cuts can be applied only to a limited set of energy functions, the two steps in solving a problem discussed earlier can no longer be separated. The energy function that we design for the scene reconstruction problem is chosen in a way that allows its efficient minimization via graph cuts. This design was greatly facilitated by another result presented in this thesis: we give a partial characterization of the class of energy functions of binary variables which can be minimized using graph cuts.

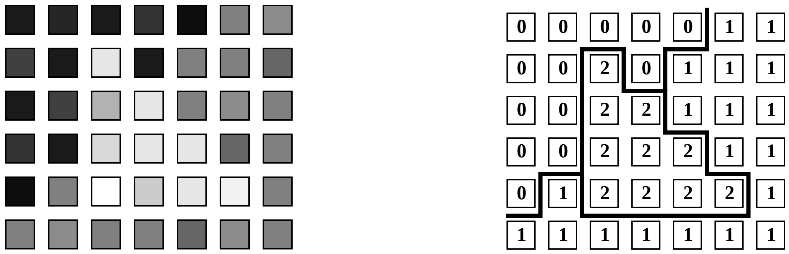
1.2 Overview of graph cut methods in vision

Greig et. al. [GPS89] were first to discover that powerful min-cut/max-flow algorithms from combinatorial optimization can be used to minimize certain important energy functions in vision. The energies addressed by Greig et. al. and by most later graph based methods (e.g. [RC98, IG98a, IG98b, BVZ98, BT99, BVZ01, BH99, Vek00, KFT+00, SVZ00, TBRN00, BJ01, KZ01, KZ02a, Zal02, BGCM02, LT03, GM03, KZG03, KSE+03, BK03]) can be represented as a posterior energy in the well-known MAP-MRF[1] framework [GG84, Li95],

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{\{p,q\}}(f_p, f_q), \tag{1.1}$$

where $f = \{f_p \ | p \in \mathcal{P}\}$ is a labeling of image $\mathcal{P}$, $D_p(\cdot)$ is a data penalty function, $V_{\{p,q\}}$ is an interaction potential, and $\mathcal{N}$ is a set of all pairs of neighboring pixels. An example of image labeling is shown in Figure 1.1. Typically, data penalties $D_p(\cdot)$ indicate individual label-preferences of pixels based on observed

[1]MAP-MRF stands for maximum *a posteriori* estimation of a Markov Random Field.



(a) An image (b) A labeling

Figure 1.1: An example of image labeling. An image in (a) is a set of pixels $\mathcal{P}$ with observed intensities I_p for each $p \in \mathcal{P}$. A labeling f shown in (b) assigns some label $f_p \in \{0, 1, 2\}$ to each pixel $p \in \mathcal{P}$. Such labels can represent depth (in stereo), object index (in segmentation), original intensity (in image restoration), or other pixel properties. Normally, graph based methods assume that the set of possible labels at each pixel is finite. Thick lines in (b) show labeling discontinuities between neighboring pixels.

intensities and pre-specified likelihood function. Interaction potentials $V_{\{p,q\}}$ encourage spatial coherence by penalizing discontinuities between neighboring pixels. The papers above show that, to date, graph based energy minimization methods provide arguably some of the most accurate solutions for the specified applications. For example, consider two recent evaluations of stereo algorithms using real imagery with dense ground truth [SZ99, SS02]. Results from [SS02] are shown in Figure 1.2.

Greig et.al. constructed a two terminal graph such that the minimum cost cut of the graph gives a globally optimal labeling f in case when the number of labels is two. Previously, exact minimization of energies like (1.1) was not possible and such energies were approached mainly with iterative algorithms like simulated

algorithm	Tsukuba			Sawtooth			Venus			Map	
	all	untex	disc	all	untex	disc	all	untex	disc	all	disc
*Layered	1.58 4	1.06 6	8.82 5	0.34 1	0.00 1	3.35 1	1.52 8	2.96 17	2.26 2	0.37 10	5.24 10
*alg. I	1.85 7	1.94 12	6.99 4	0.62 5	0.00 1	6.86 9	1.21 6	1.96 8	5.71 6	0.31 7	4.34 9
Belief prop.	1.15 1	0.42 2	6.31 1	0.98 7	0.30 12	4.83 5	1.00 4	0.76 4	9.13 11	0.84 17	5.27 11
*alg. II	1.19 2	0.23 1	6.71 2	0.73 6	0.11 7	5.71 7	1.64 11	2.75 15	5.41 5	0.61 13	6.05 12
Impr. Coop.	1.67 5	0.77 4	9.67 8	1.21 11	0.17 10	6.90 10	1.04 5	1.07 5	13.7 16	0.29 5	3.65 6
*Disc. pres.	1.78 6	1.22 8	9.71 9	1.17 9	0.08 6	5.55 6	1.61 10	2.25 11	9.06 10	0.32 8	3.33 5
*alg. II	1.27 3	0.43 3	6.90 3	0.36 2	0.00 1	3.65 2	2.79 19	5.39 20	2.54 1	1.79 20	10.1 19
*Graph cuts	1.94 9	1.09 7	9.49 7	1.30 13	0.06 5	6.34 8	1.79 14	2.61 14	6.91 7	0.31 6	3.88 7
Symbiotic	2.87 12	1.71 10	11.9 10	1.04 8	0.13 8	7.32 12	0.51 2	0.23 2	7.88 8	0.50 12	6.54 13
Var. win.	2.35 10	1.65 9	12.2 12	1.28 12	0.23 11	7.09 11	1.23 7	1.16 6	13.4 15	0.24 3	2.98 3
Adapt.Fuzzy	2.86 11	1.87 11	13.8 16	1.21 10	0.02 4	7.72 13	0.40 1	0.11 1	4.11 3	0.68 15	8.89 16
*Graph cuts	1.86 8	1.00 5	9.35 6	0.42 3	0.14 9	3.76 3	1.69 13	2.30 12	5.40 4	2.39 23	9.35 17
*Multiw.cut	8.08 24	6.53 21	25.3 25	0.61 4	0.46 15	4.60 4	0.53 3	0.31 3	8.06 9	0.26 4	3.27 4
Comp. win.	3.36 15	3.54 15	12.9 15	1.61 16	0.45 14	7.87 14	1.67 12	2.18 9	13.2 14	0.33 9	3.94 8
Realtime	4.25 19	4.47 19	15.1 20	1.32 14	0.35 13	9.21 15	1.53 9	1.80 7	12.3 12	0.81 16	11.4 22
Cooperative	3.49 16	3.65 16	14.8 18	2.03 17	2.29 21	13.4 20	2.57 18	3.52 18	26.4 24	0.22 2	2.37 1
Bay. diff.	6.49 23	11.6 26	12.3 13	1.45 15	0.72 16	9.29 16	4.00 21	7.21 23	18.4 20	0.20 1	2.49 2
Stock. diff.	3.95 17	4.08 18	15.5 22	2.45 21	0.90 18	10.6 17	2.45 16	2.41 13	21.8 22	1.31 19	7.79 15
Genetic	2.96 13	2.66 14	15.0 19	2.21 19	2.76 23	14.0 21	2.49 17	2.89 16	23.0 23	1.04 18	10.9 21
SSD+MF	5.23 22	3.80 17	24.7 24	2.21 18	0.72 17	14.0 22	3.74 20	6.82 22	12.9 13	0.66 14	9.35 17
*Max flow	2.98 14	2.00 13	15.1 21	3.47 22	3.00 24	14.2 23	2.16 15	2.24 10	21.7 21	3.13 24	16.0 25
Pix-to-pix	5.12 21	7.06 24	14.6 17	2.31 20	1.79 19	14.9 24	6.30 24	11.4 25	14.6 17	0.50 11	6.83 14
Scanl. opt.	5.08 20	6.78 22	11.9 11	4.06 23	2.64 22	11.9 18	9.44 26	14.6 26	18.2 19	1.84 21	10.2 20
Dyn. prog.	4.12 18	4.63 20	12.4 14	4.84 26	3.71 26	13.3 19	10.1 27	15.0 27	17.1 18	3.33 25	14.0 24
Shao	9.67 25	7.04 23	35.6 26	4.25 24	3.19 25	30.1 27	6.01 23	6.70 21	43.9 27	2.36 22	33.0 27
MMHM	9.76 26	13.9 27	24.4 23	4.76 25	1.87 20	22.5 25	6.48 25	10.4 24	31.3 25	8.42 27	12.7 23
Max. surf.	11.1 27	10.7 25	42.0 27	5.51 27	5.56 27	27.4 26	4.36 22	4.78 19	41.1 26	4.17 26	27.9 26

Figure 1.2: Evaluation of stereo algorithms [SS02] available at http://www.middlebury.edu/stereo. Large numbers are error percentages, small numbers are ranks of each algorithm in each column. Further details are given in section 4.5.4. Algorithms are roughly ranked according to their overall performance. Methods that use graph cuts are marked with a star. Algorithms “alg. I” and “alg. II” (#2 and #4) are our algorithms described in chapter 4.

annealing. In fact, Greig et.al. used their result to show that in practice simulated annealing reaches solutions very far from the global minimum even in a very simple example of binary image restoration.

Unfortunately, the graph cut technique in Greig et.al. remained unnoticed for almost 10 years mainly because binary image restoration looked very limited as an application. In the late 90's new computer vision techniques appeared that figured how to use min-cut/max-flow algorithms on graphs for more interesting non-binary problems. [RC98] was the first to use these algorithms to compute multi-camera stereo. Later, [IG98b, BVZ98] showed that with the right edge weights on a similar to [RC98] graph one can minimize a non-binary case of (1.1) with integer labels and linear interaction penalties. This graph construction was further generalized to handle convex smoothness terms in [Ish03]. The results in [BVZ98, BVZ01] showed that iteratively running min-cut/max-flow algorithms on appropriate graphs can be used to find provably good approximate solutions for even more general multi-label case when interaction penalties are *metrics*.

A growing number of publications in vision use graph based energy minimization techniques for applications like image segmentation [IG98b, Vek00, KFT$^+$00, BJ01, BK03], image restoration [GPS89], stereo reconstruction [RC98, IG98a, BVZ98, BT99, KZ01, KZ02a, BGCM02, LT03, GM03, KZG03], voxel occupancy [SVZ00], object recognition [BH99], augmented reality [TBRN00], texture synthesis [KSE$^+$03] and others.

1.3 Contributions and prior publications

The main goal of this thesis is applying graph based methods to the problem of reconstructing an object's 3-dimensional shape from images taken from multiple viewpoints. This is a classic vision problem. In the last few years, there has been a great deal of interest in it, partly due to a number of new applications both in vision and in graphics that require good reconstructions. While the problem can be viewed as a natural generalization of stereo, it is considerably more difficult. The major reason for this is the difficulty of reasoning about visibility. In stereo matching, most scene elements are visible from both cameras, and it is possible to obtain good results without addressing visibility constraints. In the more general scene reconstruction problem, however, very few scene elements are visible from every camera, and so the issue of visibility cannot be ignored.

We develop new algorithms for this problem with several important properties: they treat the input images symmetrically, handle visibility properly, and impose spatial smoothness while also preserving discontinuities. We give an energy minimization formulation of the multi-camera scene reconstruction problem and show how our energy function can be efficiently minimized via graph cuts.

We also address several issues common to many other vision algorithms that use graph cuts. For an important class of energy function we provide a simple criterion for determining whether a given function can be minimized via graph cuts. If possible, we show how to construct a graph for its minimization. We use this result for approximate minimization of the energy function that we introduce for the multi-camera reconstruction problem.

Finally, we present a new maxflow algorithm for computing minimum s-t cuts that outperforms other standard algorithms for many vision applications, as we demonstrate experimentally. It speeds up many graph-based methods, including the algorithms presented in this thesis.

Thus, the main contributions of this theses are:

(1) A partial characterization of the class of energy functions that can be minimized via graph cuts, together with a general-purpose construction to minimize any energy function in this class.

(2) Several new graph based algorithms for the multicamera reconstruction problem.

(3) New min cut/max flow algorithm which outperforms other standard algorithms for many vision applications.

Most of the work presented in this thesis was previously published in [KZ02b, KZ03] (contribution 1), [KZ01, KZ02a, KZG03] (contribution 2) and [BK01] (contribution 3).

1.4 Outline

The rest of the thesis is organized as follows. In chapter 2 we give some background on graphs and graph cuts, describe the two-camera stereo problem and the expansion move algorithm of [BVZ01] for solving this problem. In chapter 3 we give a partial characterization of the class of energy functions that can be minimized via graph cuts. In chapter 4 we present several new algorithms for multi-camera reconstruction problem and give experimental results. In chapter 5 we present our new min cut/max flow algorithm and experimentally demonstrate that it outperforms other standard algorithms for many vision applications. We summarize our contributions and state some open questions in chapter 6.

Chapter 2

Background and previous work

2.1 The two-camera stereo problem

Stereo matching is a fundamental problem in vision. It is routinely solved by the human brain, allowing us to perceive depth. However, the problem proved to be extremely difficult for computers to solve.

In the classical stereo vision problem, there are two cameras observing a static scene (i.e., where nothing is moving). The relative coordinate systems of the two cameras are assumed to be known, or are constrained in some fashion. The idea underlying stereopsis is to determine a correspondence (or matching) between each location p of the first image and some location q of the second image. In other words, to find the pairs of points p and q that result from the projection of the same point P into the two images. The disparity, or difference in image location, of q and p then indicates the distance from the cameras to the point P in the world. Note that a given point P need not be visible in both images — there may be some other point in the scene that hides P from view in the left or right image, causing there to be no correspondence. This situation is known as an *occlusion*.

In a standard stereo setup cameras are aligned, i.e. their positions differ only by a shift in the horizontal direction (Fig. 2.1). The two cameras are then called "left" and "right". In this setup images are *rectified*: all pairs of corresponding pixels p and q lie on the same scanline, i.e. $p_y = q_y$. A horizontal disparity $q_x - p_x$ is

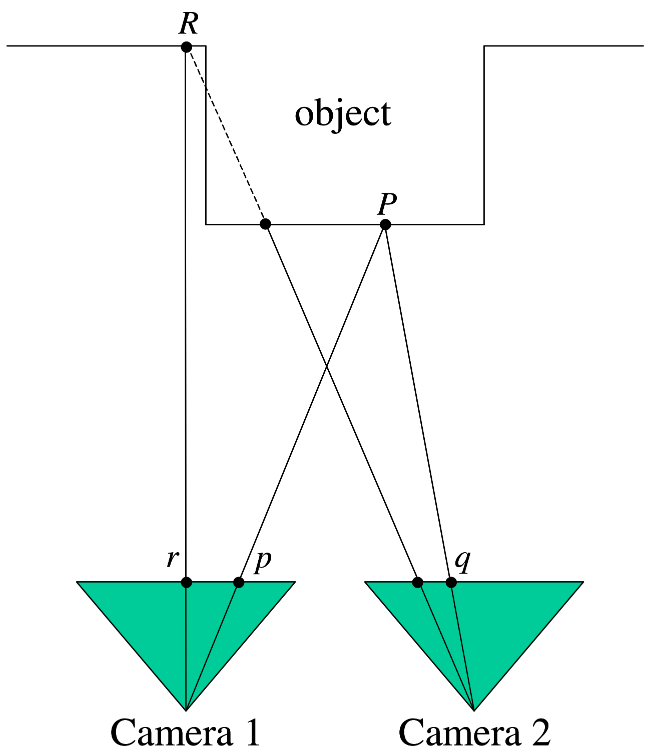


Figure 2.1: Example stereo setup with aligned cameras. The disparity $q_x - p_x$ is inversely proportional to the distance from cameras to the point P. The pixel r is occluded: the point R is visible in the left image, but not in the right.

inversely proportional to the distance to the object. Therefore, the 3-dimensional shape of the object can be computed by finding correspondences between pixels of the left and the right images.

A natural generalization of two-camera stereo is the multi-camera stereo problem. The goal is to reconstruct the 3-dimensional shape of the object from images taken by several cameras. This problem is considered in more detail in chapter 4.

2.2 Solving stereo via pixel labeling

The two-camera stereo problem is often formulated as a pixel labeling problem. One of the images is chosen to be the *primary* image and the other one the *secondary* image. Let $\mathcal{P}$ be the set of pixels in the primary image. The goal is to assign a label f_p to each pixel p in $\mathcal{P}$. This label indicates a disparity for pixel p. Therefore, in a labeling $f = \{f_p | p \in \mathcal{P}\}$ the pixel p in the primary image corresponds to pixel $p + f_p$ in the secondary image.

To compute this labeling, two assumptions are usually made:

(1) Corresponding pixels p in the primary image and $p + f_p$ in the secondary image have similar intensities.

(2) The correct disparity map f is smooth, i.e. nearby pixels p and q should have similar disparities f_p and f_q.

These two assumptions can be incorporated in the energy function of the form given in equation 1.1:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \sum_{\{p,q\} \in \mathcal{N}} V_{\{p,q\}}(f_p, f_q)$$

A typical choice for function D_p is

$$D_p(f_p) = (Intensity_1(p) - Intensity_2(p + f_p))^2$$

Several choices used as smoothness term $V_{\{p,q\}}$ are discussed in the next section, as well as techniques for minimizing energy function 1.1 with these terms.

To conclude this section, we mention some disadvantages of this formulation. First of all, it does not encode all the constraints of the visual correspondence problem. One such constraint is a *uniqueness* constraint introduced in [MP76]: each pixel can correspond to at most one pixel in the other image[1]. In the pixel labeling framework, however, it can easily happen that two pixels in the primary image map to the same pixel in the secondary image. Modeling occlusions is also problematic in this framework: it tries to assign disparities to *all* pixels in the primary image, while some of them do not have a corresponding pixel. Of course, it is possible to introduce a special label "occluded"; however, it still does not lead to unique labelings. Our experimental results in chapter 4 show that this approach does not identify occlusions very well.

Another (related) disadvantage of this formulation is that it does not treat the two input images symmetrically, which also indicates that this framework does not suit the visual correspondence problem.

2.3 Different smoothness term

An important issue is the choice of smoothness terms $V_{\{p,q\}}(\cdot,\cdot)$. For simplicity we will often refer to it as just V if this does not create confusion. One possible choice

[1]As usual in stereo, we will ignore a discretization issue. If pixel p corresponds to the middle point between adjacent pixels q and q', then it might be beneficial to mark both pairs $\{p,q\}$ and $\{p,q'\}$ as corresponding pixels.

is the L_1 distance (assuming that labels, i.e. disparities, are consecutive integers):

$$V(f_p, f_q) = \lambda \cdot |f_p - f_q|$$

An advantage of this smoothness term is that it is possible to compute efficiently a global minimum of energy function 1.1 using either of two almost identical methods developed in [IG98b, BVZ98]. This is done by computing a minimum *s*-*t* cut on a specially constructed graph. In fact, both of this methods use graphs that are very similar the one introduced in [RC98]. The same technique can also be used in a more general case when labels are integers and term $V(f_p, f_q)$ is an arbitrary convex function of $(f_p - f_q)$ [Ish03]. One special case (namely when the smoothness term is linear and data term is of the form $D_p(f_p) = |f_p - c_p|$ for constant c_p) can be solved much faster, as shown in [Zal02].

Unfortunately, convex smoothness terms have significant disadvantages. They usually work well when depth varies smoothly, however they have problems at object boundaries (discontinuities). Indeed, at the borders of objects, adjacent pixels often should have very different labels, which convex smoothness terms would give a very high penalty. As a result, the optimal labeling most likely will not be allowed to have such a large change in labels between neighboring pixels. The labelings one gets using a convex smoothness term tend to be over-smoothed around discontinuities, as demonstrated in [Vek99], for example.

To get good results at discontinuities, it is important that E not over-penalize labelings with large label changes between neighboring pixels. This requires that V be a non-convex function of $|f_p - f_q|$. An energy function with such smoothness term is called *discontinuity-preserving*. One of the simplest examples is

$$V(f_p, f_q) = \lambda \cdot T(f_p \neq f_q)$$

(Here and throughout the thesis $T(\cdot)$ denotes the indicator function: it is 1 if its argument is true and 0 otherwise). This form of smoothness term is usually called the *Potts* model.

Unfortunately, minimizing such an energy function is known to be an NP-hard problem [BVZ01], so there almost certainly are no efficient global energy minimization algorithms. However, graph cut algorithms have been developed that compute a local minimum in a strong sense [BVZ01]. These methods minimize an energy function with non-binary variables by repeatedly minimizing an energy function with binary variables.[2] In the next section we will review one of such algorithms.

2.4 The expansion move algorithm

One of the most effective algorithms for minimizing discontinuity-preserving energy functions is the expansion move algorithm, which was introduced in [BVZ99, BVZ01]. This algorithm can be used whenever the smoothness term $V(\cdot,\cdot)$ is a metric on the space of labels; this includes several important discontinuity-preserving energy functions including the Potts case (see [BVZ01] for more details).

Consider a labeling f and a particular label α. Another labeling f' is defined to be an α-expansion move from f if every pixel either keeps its old label (i.e. $f'_p = f_p$), or switches to α ($f'_p = \alpha$). This means that the set of pixels assigned the label α has increased when going from f to f'. An example of an expansion move is shown in figure 2.2.

The expansion move algorithm cycles through the labels α in some order (fixed

[2]This is somewhat analogous to the $\mathcal{Z}\pi M$ algorithm [DL02], although that method produces a global minimum of a function with non-binary variables by repeatedly minimizing a function with binary variables.

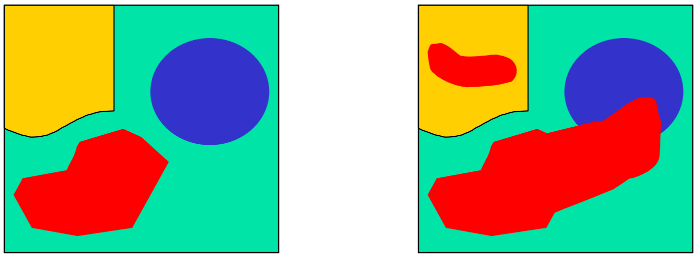


Figure 2.2: An example of an expansion move. Different colors represent different labels. The labeling on the right is a red-expansion move from the labeling on the left.

or random) and finds the lowest energy α-expansion move from the current labeling. If this expansion move has lower energy than the current labeling, then it becomes the current labeling. The algorithm terminates with a labeling that is a local minimum of the energy with respect to expansion moves; more precisely, there is no α-expansion move, for any label α, with lower energy. It is also possible to prove that such a local minimum lies within a multiplicative factor of the global minimum [BVZ01]. The factor depends on the smoothness term used; for the Potts model, for example, the factor is 2.

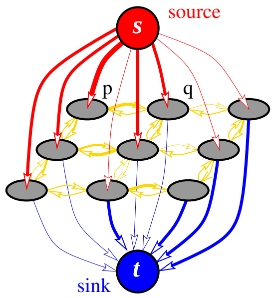
Clearly, the critical step of this algorithm is computing an expansion move with the lowest energy. Note that the search space is exponentially large. [BVZ01] showed that an optimal expansion move can be found by computing a minimum s-t cut on a specially constructed graph. Details of this construction can be found in [BVZ01]. Note, however, that they are fairly complex. In chapter 3 we will give an equivalent construction which has some advantages.

The α expansion concept introduced in [BVZ98, BVZ01] has been successfully used in many other vision and graphics algorithms [BT99, KZ01, KZ02a, LT03, KZG03, KSE[+]03]. In particular, it plays a critical role for scene reconstruction algorithms presented in this thesis.

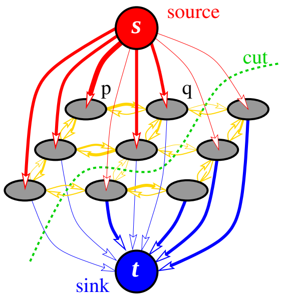
2.5 Background on graphs

In this section we review some basic facts about graphs in the context of energy minimization methods in vision. A directed weighted (capacitated) graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ consists of a set of nodes $\mathcal{V}$ and a set of directed edges $\mathcal{E}$ that connect them. Usually the nodes correspond to pixels, voxels, or other features. A graph normally contains some additional special nodes that are called terminals. In the context of vision, terminals correspond to the set of labels that can be assigned to pixels. We will concentrate on the case of graphs with two terminals. Then the terminals are usually called the *source*, s, and the *sink*, t. In Figure 2.3(a) we show a simple example of a two terminal graph (due to Greig et al. [GPS89]) that can be used to minimize the energy (1.1) using the Potts model V on a 3×3 image with two labels. There is some variation in the structure of graphs used in other energy minimization methods in vision. However, most of them are based on regular 2D or 3D grid graphs as the one in Figure 2.3(a). This is a simple consequence of the fact that normally graph nodes represent regular image pixels or voxels.

All edges in the graph are assigned some weight or cost. A cost of a directed edge (p, q) may differ from the cost of the reverse edge (q, p). In fact, ability to assign different edge weights for (p, q) and (q, p) is important for many graph-based applications in vision. Normally, there are two types of edges in the graph: n-links and t-links. N-links connect pairs of neighboring pixels or voxels. Thus, they represent a neighborhood system in the image. The cost of the n-links corresponds to a penalty for discontinuities between pixels. These costs are usually derived



(a) A graph $\mathcal{G}$



(b) A cut on $\mathcal{G}$

Figure 2.3: Example of a directed capacitated graph. Edge costs are reflected by their thickness. A similar graph-cut construction was first used in vision by Greig et al. [GPS89] for binary image restoration.

from the pixel interaction term V in the energy (1.1). T-links connect pixels with terminals (labels). The cost of a t-link connecting a pixel and a terminal corresponds to a penalty for assigning the corresponding label to the pixel. This cost is normally derived from the data term D_p in the energy function (1.1).

2.5.1 Min-cut and max-flow problems

An s-t cut C on a graph with two terminals is a partition of the nodes in the graph into two disjoint subsets $\mathcal{S}$ and $\mathcal{T}$ such that the source s is in $\mathcal{S}$ and the sink t is in $\mathcal{T}$. For simplicity, throughout this thesis we refer to s-t cuts as just *cuts*. Figure 2.3(b) shows one example of a cut. In combinatorial optimization the cost of a cut $C = \mathcal{S}, \mathcal{T}$ is defined as the sum of the costs of "boundary" edges (p, q) where $p \in \mathcal{S}$ and $q \in \mathcal{T}$. Note that cut cost is "directed" as it sums up weights of directed edges specifically from $\mathcal{S}$ to $\mathcal{T}$. The *minimum cut* problem on a graph is to find a cut that has the minimum cost among all cuts.

One of the fundamental results in combinatorial optimization is that the minimum s-t cut problem can be solved by finding a *maximum flow* from the source s to the sink t. Loosely speaking, maximum flow is the maximum "amount of water" that can be sent from the source to the sink by interpreting graph edges as directed "pipes" with capacities equal to edge weights. The theorem of Ford and Fulkerson [FF62] states that a maximum flow from s to t saturates a set of edges in the graph dividing the nodes into two disjoint parts $\mathcal{S}, \mathcal{T}$ corresponding to a minimum cut. Thus, a minimum cut can be computed from a maximum flow. In fact, the maximum flow value is equal to the cost of the minimum cut.

We can also intuitively show how min-cut (or max-flow) on a graph may help with energy minimization over image labelings. Consider an example in Figure 2.3. The graph corresponds to a 3×3 image. Any s-t cut partitions the nodes into disjoint groups each containing exactly one terminal. Therefore, any cut corresponds to some assignment of pixels (nodes) to labels (terminals). If edge weights are appropriately set based on parameters of an energy, a minimum cost cut will correspond to a labeling with the minimum value of this energy.[3]

[3]Different graph-based energy minimization methods may use different graph constructions, as well as, different rules for converting graph cuts into image labelings. Details for each method are described in the original publications.

Chapter 3

Energy functions and graph cuts

3.1 Introduction

In the last few years, several new algorithms based on graph cuts have been developed to solve energy minimization problems in computer vision (see chapters 1 and 2). Minimizing an energy function via graph cuts, however, remains a technically difficult problem. Each paper constructs its own graph specifically for its individual energy function, and in some of these cases (especially [BVZ01, KZ01, KZ02a]) the construction is fairly complex. One consequence is that researchers sometimes use heuristic methods for optimization, even in situations where the exact global minimum can be computed via graph cuts. The goal of this chapter is to precisely characterize the class of energy functions that can be minimized via graph cuts, and to give a general-purpose graph construction that minimizes any energy function in this class. Our results play a key role for scene reconstruction algorithms presented in chapter 4, provide a significant generalization of the energy minimization methods used in [BT99, BJ01, BVZ01, GPS89, KFT$^+$00, LT03, SVZ00], and show how to minimize an interesting new class of energy functions.

In this chapter we only consider energy functions involving binary-valued variables. At first glance this restriction seems severe, since most work with graph cuts considers energy functions with variables that have many possible values. For example, the algorithms presented in [BVZ01] use graph cuts to address the standard pixel labeling problem that arises in early vision, including stereo, motion and image restoration. In this problem the variables represent individual pixels, and the possible values for an individual variable represent, e.g., its possible displacements or intensities. However, many of the graph cut methods that handle multiple possible values work by repeatedly minimizing an energy function involving only binary variables.

Consider, for example, the expansion move algorithm of [BVZ01] described in section 2.4. In the α-expansion operation each pixel makes a binary decision: either to keep its old label, or to switch to the new label α. Thus, computing an optimal α-expansion move can be viewed as minimizing an energy function of binary variables with one binary variable per pixel.

As we will see, our results generalize the algorithms given in [BT99, BJ01, BVZ01, GPS89, KFT+00, SVZ00], and can be easily applied to problems like pixel-labeling, even though the pixels have many possible labels.

3.1.1 Summary of our results

In this chapter we focus on two classes of energy functions. Let $\{x_1, \ldots, x_n\}$, $x_i \in \{0, 1\}$ be a set of binary-valued variables. We define the class $\mathcal{F}^2$ to be functions that can be written as a sum of functions of up to 2 binary variables at a time,

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i, x_j). \tag{3.1}$$

We define the class $\mathcal{F}^3$ to be functions that can be written as a sum of functions of up to 3 binary variables at a time,

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i, x_j) + \sum_{i<j<k} E^{i,j,k}(x_i, x_j, x_k). \tag{3.2}$$

Obviously, the class $\mathcal{F}^2$ is a strict subset of the class $\mathcal{F}^3$. However, to simplify the discussion we will begin by focusing on $\mathcal{F}^2$. Note that there is no restriction on the signs of the energy functions or of the individual terms.

The main results in this chapter are:

- A precise characterization of the class of functions in $\mathcal{F}^3$ that can be minimized using graph cuts.
- A general-purpose graph construction for minimizing any function in this class.
- A necessary condition for any function of binary variables to be minimized via graph cuts.

The rest of the chapter is organized as follows. We discuss the importance of energy functions with binary variables in the context of the expansion move algorithm in section 3.1.2. In section 3.2 we formalize the relationship between graphs and energy functions, and provide a precise definition of the problem that we wish to solve. Section 3.3 contains our main theorem for the class of functions $\mathcal{F}^2$ and shows how this result can be used for both stereo and multi-camera scene reconstruction. Section 3.4 gives our results for the broader class $\mathcal{F}^3$, and shows how this result can be used for multi-camera scene reconstruction. A necessary condition for an arbitrary function of binary variables to be minimized via graph cuts is presented in section 3.5. We discuss some related work in the theory of submodular functions in section 3.6, and give a summary of our graph constructions in section 3.7. An NP-hardness result and a proof of one of our theorems are deferred to section 3.8.

3.1.2 The expansion move algorithm

The key sub-problem in the expansion move algorithm is to compute the lowest energy labeling within a single α-expansion of f. This sub-problem is solved efficiently with a single graph cut [BVZ01], using a somewhat intricate graph construction.

It is important to note that this sub-problem is an energy minimization problem over binary variables, even though the overall problem that the expansion move algorithm is solving involves non-binary variables. This is because each pixel will either keep its old value under f or acquire the new label α.

Formally, any labeling f' within a single α-expansion of the initial labeling f can be encoded by a binary vector $x = \{x_p | p \in \mathcal{P}\}$ where $f'(p) = f(p)$ if $x_p = 0$, and $f'(p) = \alpha$ if $x_p = 1$. Let us denote the labeling defined by a binary vector x as f^x. Since the energy function E is defined over all labelings, it is obviously also defined over labelings specified by binary vectors. The key step in the expansion move algorithm is therefore to find the minimum of $E(f^x)$ over all binary vectors x.

The importance of energy functions of binary variables does not arise simply from the expansion move algorithm. Instead, it results from the fact that a graph cut effectively assigns one of two possible values to each vertex of the graph. So in a certain sense any energy minimization construction based on graph cuts relies on intermediate binary variables.

It is, of course, possible to use graph cuts in such a manner that variables in the original problem do not correspond in a one-to-one manner with vertices in the graph. This kind of complex transformation lies at the heart of the graph cut algorithms that compute a global minimum [IG98b, Ish03]. However, the NP-hardness result given in [BVZ01] shows that (unless P=NP) such as result cannot be achieved for even the simplest discontinuity-preserving energy function.

3.2 Representing energy functions with graphs

In section 2.5 we defined the notion of graph cuts. For the purposes of this chapter we will use a slightly different view on graph cuts, which we now present.

Let us consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with terminals s and t, thus $\mathcal{V} = \{v_1, \ldots, v_n, s, t\}$. Each cut on $\mathcal{G}$ has some cost; therefore, $\mathcal{G}$ represents the energy function mapping from all cuts on $\mathcal{G}$ to the set of nonnegative real numbers. Any cut $\mathcal{S}, \mathcal{T}$ can be described by n binary variables $x_1, \ldots, x_n$ corresponding to vertices in $\mathcal{G}$ (excluding the source and the sink): $x_i = 0$ when $v_i \in \mathcal{S}$, and $x_i = 1$ when $v_i \in \mathcal{T}$. Therefore, the energy E that $\mathcal{G}$ represents can be viewed as a function of n binary variables: $E(x_1, \ldots, x_n)$ is equal to the cost of the cut defined by the configuration $x_1, \ldots, x_n$ ($x_i \in \{0, 1\}$). Note that the configuration that minimizes E will not change if we add a constant to E.

We can efficiently minimize E by computing a minimum s-t-cut on $\mathcal{G}$. This naturally leads to the question: what is the class of energy functions E for which we can construct a graph that represents E?

We can also generalize our construction. Above we used each vertex (except the source and the sink) for encoding one binary variable. Instead we can specify a subset $\mathcal{V}_0 = \{v_1, \ldots, v_k\} \subset \mathcal{V} - \{s, t\}$ and introduce variables only for the vertices in this set. Then there may be several cuts corresponding to a configuration $x_1, \ldots, x_k$. If we define the energy $E(x_1, \ldots, x_k)$ as the minimum among the costs of all such cuts, then a minimum s-t-cut on $\mathcal{G}$ will again yield a configuration which minimizes E.

We will summarize the graph constructions that we allow in the following definition.

Definition 3.1 *A function E of n binary variables is called* graph-representable *if there exists a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with terminals s and t and a subset of vertices $\mathcal{V}_0 = \{v_1, \ldots, v_n\} \subset \mathcal{V} - \{s, t\}$ 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 a minimum s-t-cut among all cuts $C = \mathcal{S}, \mathcal{T}$ in which $v_i \in \mathcal{S}$, if $x_i = 0$, and $v_i \in \mathcal{T}$, if $x_i = 1$ $(1 \leq i \leq n)$. We say that E is* exactly represented *by $\mathcal{G}$, $\mathcal{V}_0$ if this constant is zero.*

The following lemma is an obvious consequence of this definition.

Lemma 3.2 *Suppose the energy function E is graph-representable by a graph $\mathcal{G}$ and a subset $\mathcal{V}_0$. Then it is possible to find the exact minimum of E in polynomial time by computing a minimum s-t-cut on $\mathcal{G}$.*

In this chapter we will give a complete characterization of the classes $\mathcal{F}^2$ and $\mathcal{F}^3$ in terms of graph representability, and show how to construct graphs for minimizing graph-representable energies within these classes. Moreover, we will give a necessary condition for all other classes which must be met for a function to be graph-representable. Obviously, it would be sufficient to consider only the class $\mathcal{F}^3$, since $\mathcal{F}^2 \subset \mathcal{F}^3$. However, the condition for $\mathcal{F}^2$ is simpler so we will consider it separately.

Note that the energy functions we consider can be negative, as can the individual terms in the energy functions. However, the graphs that we construct must have non-negative edge weights. All previous work that used graph cuts for energy minimization dealt with non-negative energy functions and terms. As a result, in this respect our results are more general.

3.3 The class $\mathcal{F}^2$

Our main result for the class $\mathcal{F}^2$ is the following theorem.

Theorem 3.3 ($\mathcal{F}^2$ theorem) *Let E be a function of n binary variables from the class $\mathcal{F}^2$, i.e.*

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i, x_j). \tag{3.3}$$

Then E is graph-representable if and only if each term $E^{i,j}$ satisfies the inequality

$$E^{i,j}(0,0) + E^{i,j}(1,1) \leq E^{i,j}(0,1) + E^{i,j}(1,0). \tag{3.4}$$

We will call functions satisfying the condition of equation (3.4) *regular.*[1] This theorem thus states that regularity is a necessary and sufficient condition for graph-representability, at least in $\mathcal{F}^2$.

In this section we will give a constructive proof that regularity is a sufficient condition, by describing how to build a graph that represents an arbitrary regular function in $\mathcal{F}^2$. The other half of the theorem is proved in much more generality in section 3.5, where we show that a regularity is also a necessary condition.

[1]The representation of a function E as a sum in equation (3.3) is not unique. However, we will show in section 3.4 that the definition of regularity does not depend on this representation.

Regularity is thus an extremely important property, as it allows energy functions to be minimized using graph cuts. Moreover, without the regularity constraint, the problem becomes intractable. Our next theorem shows that minimizing an arbitrary non-regular function is NP-hard, even if we restrict our attention to $\mathcal{F}^2$.

Theorem 3.4 (NP-hardness) *Let E^2 be a non-regular function of two variables. Then minimizing functions of the form*

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{(i,j)\in\mathcal{N}} E^2(x_i, x_j)$$

where E^i are arbitrary functions of one variable and $\mathcal{N} \subset \{(i,j)|1 \leq i < j \leq n\}$, is NP-hard.

The proof of this theorem is deferred to section 3.8. Note that this theorem implies the intractability of minimizing the entire class of non-regular functions in $\mathcal{F}^2$. It allows for the existence of non-regular functions in $\mathcal{F}^2$ that can be minimized efficiently.[2]

3.3.1 Graph construction for $\mathcal{F}^2$

In this section we will prove the constructive part of $\mathcal{F}^2$ theorem. Let E be a regular function of the form as given in the theorem. We will construct a graph for each term separately and then "merge" all graphs together. This is justified by additivity theorem given in section 3.8.

[2] Steven Gortler and colleagues have investigated a particularly simple example, namely functions that can be made regular by interchanging the sense of 0 and 1 for some set of variables.

The graph will contain $n+2$ vertices: $\mathcal{V} = \{s, t, v_1, \ldots, v_n\}$. Each non-terminal vertex v_i will encode the binary variable x_i. For each term of E we will add one or more edges that we describe next.

First, consider a term E^i depending on one variable x_i. If $E^i(0) < E^i(1)$ then we add the edge (s, v_i) with the weight $E^i(1) - E^i(0)$ (Figure 3.1(a)). Otherwise we add the edge (v_i, t) with the weight $E^i(0) - E^i(1)$ (Figure 3.1(b)). It easy to see that in both cases the constructed graph represents the function E^i (but with different constants: $E^i(0)$ in the former case and $E^i(1)$ in the latter).

Now consider a term $E^{i,j}$ depending on two variables x_i, x_j. It is convenient to represent it as the following table:

$$E^{i,j} = \begin{array}{|c|c|}\hline E^{i,j}(0,0) & E^{i,j}(0,1) \\ \hline E^{i,j}(1,0) & E^{i,j}(1,1) \\ \hline \end{array} = \begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline \end{array}$$

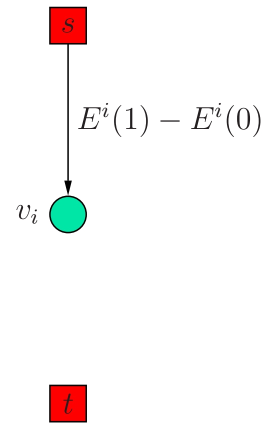
We can rewrite it as

$$\begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline \end{array} = A + \begin{array}{|c|c|}\hline 0 & 0 \\ \hline C-A & C-A \\ \hline \end{array} + \begin{array}{|c|c|}\hline 0 & D-C \\ \hline 0 & D-C \\ \hline \end{array} + \begin{array}{|c|c|}\hline 0 & B+C-A-D \\ \hline 0 & 0 \\ \hline \end{array}$$

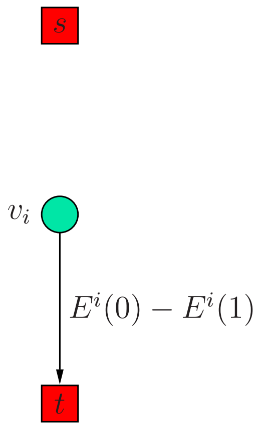
The first term is a constant function, so we don't need to add any edges for it. The second and the third terms depend only on one variable x_i and x_j, respectively. Therefore, we can use the construction given above. To represent the last term, we add an edge (v_i, v_j) with the weight $B + C - A - D$ (Figure 3.1(c)). Note that this weight is nonnegative due to the regularity of E.

A complete graph for $E^{i,j}$ will contain three edges. One possible case ($C - A > 0$ and $D - C < 0$) is illustrated in Figure 3.1(d).

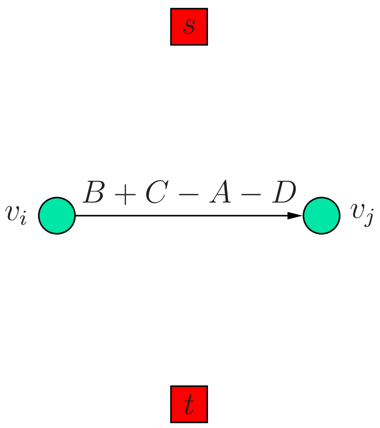
Note that we did not introduce any additional vertices for representing binary interactions of binary variables. This is in contrast to the construction in [BVZ01] which added auxiliary vertices for representing energies that we just considered.



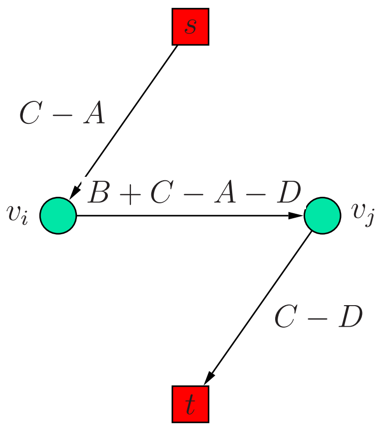
(a) Graph for E^i where $E^i(0) > E^i(1)$



(b) Graph for E^i where $E^i(0) \not> E^i(0)$



(c) Third edge for $E^{i,j}$



(d) Complete graph for $E^{i,j}$

if $C > A$ and $C > D$

Figure 3.1: Graphs for functions in $\mathcal{F}^2$

Our construction yields a smaller graph and, thus, the minimum cut can potentially be computed faster.

3.3.2 Example: applying our results for stereo problem

Our results give a generalization of a number of previous algorithms [BT99, BJ01, BVZ01, GPS89, KFT$^+$00, LT03, SVZ00] in the following sense. Each of these methods used a graph cut algorithm that was specifically constructed to minimize a certain form of energy function. The class of energy functions that we show how to minimize is much larger, and includes the techniques used in all of these methods as special cases.

To illustrate the power of our results, we return to the use of the expansion move algorithm for stereo problem described in section 2.4. Recall that the key sub-problem is to find a minimum energy labeling within a single α-expansion of f, which is equivalent to minimizing the energy over binary variables x_i.

The paper that proposed the expansion move algorithm [BVZ01] showed how to solve the key sub-problem with graph cuts as long as D_p is non-negative and V is a metric on the space of labels. This involved an elaborate graph construction and several associated theorems.

Using our results, we can recreate the proof that such an energy function can be solved in just a few lines. All that we need is to prove that E is regular if D_p is non-negative and V is a metric. Obviously, the form of D_p doesn't matter; we simply have to show that if V is a metric the individual terms satisfy the inequality in equation (3.4). Consider two neighboring pixels p, q, with associated binary variables i, j. We have

$$\begin{array}{|c|c|}\hline E^{i,j}(0,0) & E^{i,j}(0,1) \\ \hline E^{i,j}(1,0) & E^{i,j}(1,1) \\ \hline \end{array} = \begin{array}{|c|c|}\hline V(f_p, f_q) & V(f_p, \alpha) \\ \hline V(\alpha, f_q) & V(\alpha, \alpha) \\ \hline \end{array}$$

If V is a metric, by definition, for any labels α, β, γ $V(\alpha, \alpha) = 0$ and $V(\beta, \alpha) + V(\alpha, \gamma) \geq V(\beta, \gamma)$. This gives the inequality of equation (3.4) and shows that E can be minimized using graph cuts.

3.4 The class $\mathcal{F}^3$

Before stating our theorem for the class $\mathcal{F}^3$ we will extend the definition of regularity to arbitrary functions of binary variables. We will need a notion of *projections*.

Suppose we have a function E of n binary variables. If we fix m of these variables then we get a new function E' of $n-m$ binary variables; we will call this function a *projection* of E. The notation for projections is as follows.

Definition 3.5 *Let $E(x_1, \ldots, x_n)$ be a function of n binary variables, and let I, J be a disjoint partition of the set of indices $\{1, \ldots, n\}$: $I = \{i(1), \ldots, i(m)\}$, $J = \{j(1), \ldots, j(n-m)\}$. Let $\alpha_{i(1)}, \ldots, \alpha_{i(m)}$ be binary constants. A* projection *$E' = E[x_{i(1)} = \alpha_{i(1)}, \ldots, x_{i(m)} = \alpha_{i(m)}]$ is a function of $n-m$ variables defined by*

$$E'(x_{j(1)}, \ldots, x_{j(n-m)}) = E(x_1, \ldots, x_n),$$

where $x_i = \alpha_i$ for $i \in I$. We say that we fix *the variables $x_{i(1)}$, ..., $x_{i(m)}$.*

Now we can give a generalized definition of regularity.

Definition 3.6

- *All functions of one variable are regular.*

- *A function E of two variables is called regular if $E(0,0)+E(1,1) \leq E(0,1)+E(1,0)$.*

- *A function E of more than two variables is called regular if all projections of E of two variables are regular.*

For the class $\mathcal{F}^2$ this definition is equivalent to the previous one. A proof of this fact is given in section 3.4.3.

Now we are ready to formulate our main theorem for $\mathcal{F}^3$.

Theorem 3.7 ($\mathcal{F}^3$ theorem) *Let E be a function of n binary variables from $\mathcal{F}^3$, i.e.*

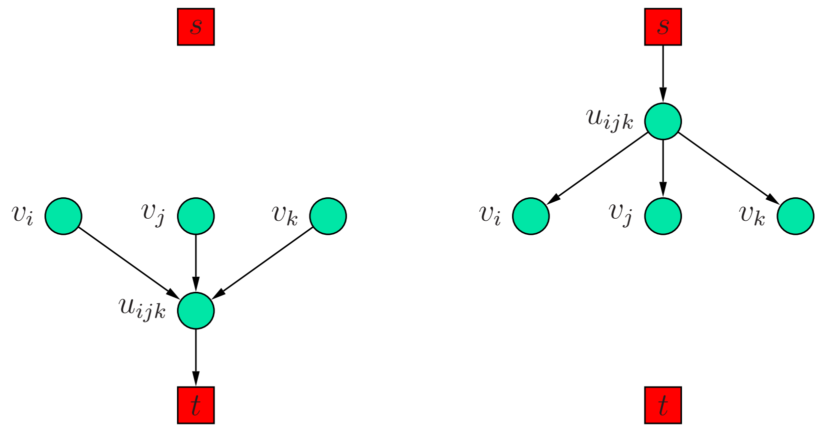
$$E(x_1,\ldots,x_n) = \sum_i E^i(x_i) + \sum_{i<j} E^{i,j}(x_i,x_j) + \sum_{i<j<k} E^{i,j,k}(x_i,x_j,x_k) \tag{3.5}$$

Then E is graph-representable if and only if E is regular.

3.4.1 Graph construction for $\mathcal{F}^3$

Consider a regular function E of the form given in equation (3.5). The regularity of E does not necessarily imply that each term in (3.5) is regular. However, we will give an algorithm for converting any regular function in $\mathcal{F}^3$ to the form (3.5) with each term regular (see regrouping theorem in the next section). Note that this was not necessary for the class $\mathcal{F}^2$: for any representation of a regular function in the form (3.3) each term in the sum is regular.

Here we will give graph construction for a regular function E of the form as in the $\mathcal{F}^3$ theorem assuming that each term of E is regular. The graph will contain



(a) Edge weights are all P (b) Edge weights are all $-P$

Figure 3.2: Graphs for functions in $\mathcal{F}^3$

$n+2$ vertices: $\mathcal{V} = \{s, t, v_1, \ldots, v_n\}$ as well as some additional vertices described below. Each non-terminal vertex v_i will encode the binary variable x_i. For each term of E we will add one or more edges and possibly an additional (unique) vertex that we describe next. Again, our construction is justified by the additivity lemma given in section 3.8.

Terms depending on one and two variables were considered in the previous section, so we will concentrate on a term $E^{i,j,k}$ depending on three variables x_i, x_j, x_k. It is convenient to represent it as the following table:

$$E^{i,j,k} = \begin{array}{|c|c|} \hline E^{i,j}(0,0,0) & E^{i,j}(0,0,1) \\ \hline E^{i,j}(0,1,0) & E^{i,j}(0,1,1) \\ \hline E^{i,j}(1,0,0) & E^{i,j}(1,0,1) \\ \hline E^{i,j}(1,1,0) & E^{i,j}(1,1,1) \\ \hline \end{array} = \begin{array}{|c|c|} \hline A & B \\ \hline C & D \\ \hline E & F \\ \hline G & H \\ \hline \end{array}$$

Let us denote $P = (A+D+F+G)-(B+C+E+H)$. We consider two cases:

Case 1: $P >= 0$. We can rewrite $E^{i,j,k}$ as

$$
\begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline E & F \\ \hline G & H \\ \hline\end{array}
= A +
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline P_1 & P_1 \\ \hline P_1 & P_1 \\ \hline\end{array}
+
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline P_2 & P_2 \\ \hline 0 & 0 \\ \hline P_2 & P_2 \\ \hline\end{array}
+
\begin{array}{|c|c|}\hline 0 & P_3 \\ \hline 0 & P_3 \\ \hline 0 & P_3 \\ \hline 0 & P_3 \\ \hline\end{array}
+
$$

$$
+
\begin{array}{|c|c|}\hline 0 & P_{23} \\ \hline 0 & 0 \\ \hline 0 & P_{23} \\ \hline 0 & 0 \\ \hline\end{array}
+
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline P_{31} & 0 \\ \hline P_{31} & 0 \\ \hline\end{array}
+
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline P_{12} & P_{12} \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline\end{array}
+
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & -P \\ \hline\end{array}
$$

where $P_1 = F - B$, $P_2 = G - E$, $P_3 = D - C$, $P_{23} = B + C - A - D$, $P_{31} = B + E - A - F$, $P_{12} = C + E - A - G$. $E^{i,j,k}$ is regular, therefore by considering projections $E^{i,j,k}[x_1 = 0]$, $E^{i,j,k}[x_2 = 0]$, $E^{i,j,k}[x_3 = 0]$ we conclude that P_{23}, P_{31}, P_{12} are non-negative.

The first term is a constant function, so we don't need to add any edges for it. The next three terms depend only on one variable (x_i, x_j, x_k, respectively) so we can use the construction given in the previous section. The next three terms depend on two variables; non-negativity of P_{23}, P_{31}, P_{12} implies that they are all regular. Therefore, we can again use the construction of the previous section. (Three edges will be added: (v_j, v_k), (v_k, v_i), (v_i, v_j) with weights P_{23}, P_{31}, P_{12}, respectively).

To represent the last term, we will add an auxiliary vertex u_{ijk} and four edges (v_i, u_{ijk}), (v_j, u_{ijk}), (v_k, u_{ijk}), (u_{ijk}, t) with the weight P. This is shown in figure 3.2(a). Let us prove these four edges exactly represent the function

$$\begin{array}{|c|c|}\hline P & P \\ \hline P & P \\ \hline P & P \\ \hline P & 0 \\ \hline \end{array} = P + \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & -P \\ \hline \end{array}$$

If $x_i = x_j = x_k = 1$ then the cost of the minimum cut is 0 (the minimum cut is $\mathcal{S} = \{s\}$, $\mathcal{T} = \{v_i, v_j, v_k, u_{ijk}, t\}$. Suppose at least one of the variables x_i, x_j, x_k is 0; without loss of generality, we can assume that $x_i = 0$, i.e. $v_i \in \mathcal{S}$. If $u_{ijk} \in \mathcal{S}$ then the edge (u_{ijk}, t) is cut; if $u_{ijk} \in \mathcal{T}$ the edge (v_i, u_{ijk}) is cut yielding the cost P. Hence, the cost of a minimum cut is at least P. However, if $u_{ijk} \in \mathcal{S}$ the cost of the cut is exactly P no matter where the vertices v_i, v_j, v_k are. Therefore, the cost of a minimum cut will be 0 if $x_i = x_j = x_k = 1$ and P otherwise, as desired.

Case 2: $P < 0$. This case is similar to the case 1. We can rewrite $E^{i,j,k}$ as

$$\begin{array}{|c|c|}\hline A & B \\ \hline C & D \\ \hline E & F \\ \hline G & H \\ \hline \end{array} = H + \begin{array}{|c|c|}\hline P_1 & P_1 \\ \hline P_1 & P_1 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|}\hline P_2 & P_2 \\ \hline 0 & 0 \\ \hline P_2 & P_2 \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|}\hline P_3 & 0 \\ \hline P_3 & 0 \\ \hline P_3 & 0 \\ \hline P_3 & 0 \\ \hline \end{array} +$$

$$+ \begin{array}{|c|c|}\hline 0 & 0 \\ \hline P_{32} & 0 \\ \hline 0 & 0 \\ \hline P_{32} & 0 \\ \hline \end{array} + \begin{array}{|c|c|}\hline 0 & P_{13} \\ \hline 0 & P_{13} \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline P_{21} & P_{21} \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|}\hline P & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$$

where $P_1 = C - G$, $P_2 = B - D$, $P_3 = E - F$, $P_{32} = F + G - E - H$, $P_{13} = D + G - C - H$, $P_{21} = D + F - B - H$. By considering projections $E^{i,j,k}[x_1 = 1]$, $E^{i,j,k}[x_2 = 1]$, $E^{i,j,k}[x_3 = 1]$ we conclude that P_{32}, P_{13}, P_{21} are non-negative.

As in the previous case, all terms except the last are regular and depend on at most two variables, so we can use the construction of the previous section. We will have, for example, edges (v_k, v_j), (v_i, v_k), (v_j, v_i) with weights P_{32}, P_{13}, P_{21}, respectively.

To represent the last term, we will add an auxiliary vertex u_{ijk} and four edges (u_{ijk}, v_i), (u_{ijk}, v_j), (u_{ijk}, v_k), (s, u_{ijk}) with the weight $-P$. This is shown in figure 3.2(b).

Note that in both cases we added an auxiliary vertex u_{ijk}. It is easy to see that this is necessary since graphs without auxiliary vertices can only represent functions in $\mathcal{F}^2$. Each edge represents a function of at most two variables, so the whole graph represents a function that is a sum of terms of at most two variables.

3.4.2 Constructive proof of the regrouping theorem

Now we will show how to convert a regular function in $\mathcal{F}^3$ to the form (3.5) with each term regular.

Theorem 3.8 (regrouping) *Any regular function E from the class $\mathcal{F}^3$ can be rewritten as a sum of terms such that each term is regular and depends on at most three variables.*

We will assume that E given as

$$E(x_1, \ldots, x_n) = \sum_{i<j<k} E^{i,j,k}(x_i, x_j, x_k),$$

where i, j, k are indices from the set $\{1, \ldots, n\}$ (we omitted terms involving functions of one and two variables since they can be viewed as functions of three variables).

We begin by giving a definition.

Definition 3.9 *The functional π will be a mapping from the set of all functions (of binary variables) to the set of real numbers which is defined as follows. For a function $E(x_1, \ldots, x_n)$*

$$\pi(E) = \sum_{x_1 \in \{0,1\}, \ldots, x_n \in \{0,1\}} \left(\Pi_{i=1}^{n} (-1)^{x_i}\right) E(x_1, \ldots, x_n).$$

For example, for a function E of two variables $\pi(E) = E(0,0) - E(0,1) - E(1,0) + E(1,1)$. Note that a function E of two variables is regular if and only if $\pi(E) \leq 0$. (Some other properties of π are discussed at the end of this chapter.)

It is trivial to check the following lemma.

Lemma 3.10 *The functional π has the following properties.*

- *π is linear, i.e. for a scalar c and two functions E', E'' of n variables $\pi(E' + E'') = \pi(E') + \pi(E'')$ and $\pi(c \cdot E') = c \cdot \pi(E')$.*
- *If E is a function of n variables that does not depend on at least one of the variables then $\pi(E) = 0$.*

We will prove the theorem using the following lemma and a trivial induction argument.

Definition 3.11 *Let $E^{i,j,k}$ be a function of three variables. The functional $N(E^{i,j,k})$ is defined as the number of projections of two variables of $E^{i,j,k}$ with positive values of the functional π.*

Note that $N(E^{i,j,k}) = 0$ exactly when $E^{i,j,k}$ is regular.

Lemma 3.12 *Suppose the function E of n variables can be written as*

$$E(x_1, \ldots, x_n) = \sum_{i<j<k} E^{i,j,k}(x_i, x_j, x_k),$$

where some of the terms are not regular. Then it can be written as

$$E(x_1, \ldots, x_n) = \sum_{i<j<k} \tilde{E}^{i,j,k}(x_i, x_j, x_k),$$

where

$$\sum_{i<j<k} N(\tilde{E}^{i,j,k}) < \sum_{i<j<k} N(E^{i,j,k}).$$

PROOF:

For simplicity of notation let us assume that the term $E^{1,2,3}$ is not regular and $\pi(E^{1,2,3}[x_3 = 0]) > 0$ or $\pi(E^{1,2,3}[x_3 = 1]) > 0$ (we can ensure this by renaming indices). Let

$$C_k = \max_{\alpha_k \in \{0,1\}} \pi(E^{1,2,k}[x_k = \alpha_k]) \qquad k \in \{4, \ldots, n\}$$

$$C_3 = -\sum_{k=4}^{n} C_k$$

Now we will modify the terms $E^{1,2,3}$, ..., $E^{1,2,n}$ as follows:

$$\tilde{E}^{1,2,k} \equiv E^{1,2,k} - R[C_k] \qquad k \in \{3, \ldots, n\}$$

where $R[C]$ is the function of two variables x_1 and x_2 defined by the table

$$R[C] = \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & C \\ \hline \end{array}$$

(other terms are unchanged: $\tilde{E}^{i,j,k} \equiv E^{i,j,k}$, $(i,j) \neq (1,2)$). We have

$$E(x_1, \ldots, x_n) = \sum_{i<j<k} \tilde{E}^{i,j,k}(x_i, x_j, x_k)$$

since $\sum_{k=3}^{n} C_k = 0$ and $\sum_{k=3}^{n} R[C_k] \equiv 0$.

If we consider $R[C]$ as a function of n variables and take a projection of two variables where the two variables that are not fixed are x_i and x_j $(i < j)$, then the functional π will be C, if $(i,j)=(1,2)$, and 0 otherwise since in the latter case a projection actually depends on at most one variable. Hence, the only projections of two variables that could have changed their value of the functional π are $\tilde{E}^{1,2,k}[x_3 = \alpha_3, \ldots, x_n = \alpha_n]$, $k \in \{3, \ldots, n\}$, if we treat $\tilde{E}^{1,2,k}$ as functions of n variables, or $\tilde{E}^{1,2,k}[x_k = \alpha_k]$, if we treat $\tilde{E}^{1,2,k}$ as functions of three variables.

First let us consider terms with $k \in \{4, \ldots, n\}$. We have $\pi(E^{1,2,k}[x_k = \alpha_k]) \leq C_k$, thus

$$\pi(\tilde{E}^{1,2,k}[x_k = \alpha_k]) = \pi(E^{1,2,k}[x_k = \alpha_k]) - \pi(R[C_k][x_k = \alpha_k]) \leq C_k - C_k = 0$$

Therefore we did not introduce any non-regular projections for these terms.

Now let us consider the term $\pi(\tilde{E}^{1,2,3}[x_3 = \alpha_3])$. We can write

$$\pi(\tilde{E}^{1,2,3}[x_3 = \alpha_3]) = \pi(E^{1,2,3}[x_3 = \alpha_3]) - \pi(R[C_3][x_3 = \alpha_3]) =$$
$$= \pi(E^{1,2,3}[x_3 = \alpha_3]) - (-\sum_{k=4}^{n} C_k) = \sum_{k=3}^{n} \pi(E^{1,2,k}[x_k = \alpha_k])$$

where $\alpha_k = \arg\max_{\alpha \in \{0,1\}} \pi(E^{1,2,k}[x_k = \alpha])$, $k \in \{4, \ldots, n\}$. The last expression is just $\pi(E[x_3 = \alpha_3, \ldots, x_n = \alpha_n])$ and is non-positive since E is regular by assumption. Hence, values $\pi(\tilde{E}^{1,2,3}[x_3 = 0])$ and $\pi(\tilde{E}^{1,2,3}[x_3 = 1])$ are both non-positive and, therefore, the number of non-regular projections has decreased. ■

The complexity of a single step is $O(n)$ since it involves modifying at most n terms. Therefore, the complexity of the whole algorithm is $O(mn)$ where m is the number of terms in (3.5) since in the beginning $\sum_{i<j<k} N(E^{i,j,k})$ is at most $6m$ and each step decreases it by at least one.

3.4.3 The two definitions of regularity are equivalent

We now prove that the definition of regularity 3.6 is equivalent to the previous definition of regularity for the class $\mathcal{F}^2$. Note that the latter one is formulated not only as a property of the function E but also as a property of its representation as a sum in equation (3.3). We will show equivalence for an arbitrary representation, which will imply that the definition of regularity for the class $\mathcal{F}^2$ depends only on E but not on the representation.

Let us consider a graph-representable function E from the class $\mathcal{F}^2$:

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

Consider a projection where the two variables that are not fixed are x_i and x_j. By lemma 3.10 the value of the functional π of this projection is equal to $\pi(E^{i,j})$ (all other terms yield zero). Therefore, this projection is regular if and only if $E^{i,j}(0,0) + E^{i,j}(1,1) \leq E^{i,j}(0,1) + E^{i,j}(1,0)$, which means that the two definitions are equivalent.

3.5 More general classes of energy functions

Finally, we give a necessary condition for graph representability for arbitrary functions of binary variables.

Theorem 3.13 (regularity) *Let E be a function of binary variables. If E is not regular then E is not graph-representable.*

This theorem will imply the corresponding directions of the $\mathcal{F}^2$ and $\mathcal{F}^3$ theorems (theorems 3.3 and 3.7).

Definition 3.14 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, $v_1, \ldots, v_k$ be a subset of vertices $\mathcal{V}$ and $\alpha_1, \ldots, \alpha_k$ be binary constants whose values are from $\{0, 1\}$. We will define the graph $\mathcal{G}[x_1 = \alpha_1, \ldots, x_k = \alpha_k]$ as follows. Its vertices will be the same as in $\mathcal{G}$*

and its edges will be all edges of $\mathcal{G}$ plus additional edges corresponding to vertices $v_1, \ldots, v_k$: for a vertex v_i, we add the edge (s, v_i), if $\alpha_i = 0$, or (v_i, t), if $\alpha_i = 1$, with an infinite capacity.

It should be obvious that these edges enforce the following constraints on nodes $v_1, \ldots, v_k$ in a minimum cut S,T: if $\alpha_i = 0$ then $v_i \in \mathcal{S}$, and if $\alpha_i = 1$ then $v_i \in \mathcal{T}$. (If, for example, $\alpha_i = 0$ and $v_i \in \mathcal{T}$ then the edge (s, v_i) must be cut yielding an infinite cost, so it would not a minimum cut.)

Now we can give a definition of graph representability which is equivalent to the definition 3.1. This new definition will be more convenient for the proof.

Definition 3.15 *We say that the function E of n binary variables is exactly represented by the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the set $\mathcal{V}_0 \subset \mathcal{V}$ if for any configuration $\alpha_1, \ldots, \alpha_n$ the cost of a minimum cut on $\mathcal{G}[x_1 = \alpha_1, \ldots, x_k = \alpha_k]$ is $E(\alpha_1, \ldots, \alpha_n)$.*

Lemma 3.16 *Any projection of a graph-representable function is graph-representable.*

Proof: Let E be a graph-representable function of n variables, and the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the set $\mathcal{V}_0$ represents E. Suppose that we fix variables $x_{i(1)}, \ldots, x_{i(m)}$. It is straightforward to check that the graph $\mathcal{G}[x_{i(1)} = \alpha_{i(1)}, \ldots, x_{i(m)} = \alpha_{i(m)}]$ and the set $\mathcal{V}_0' = \mathcal{V}_0 - \{v_{i(1)}, \ldots, v_{i(m)}\}$ represent the function $E' = E[x_{i(1)} = \alpha_{i(1)}, \ldots, x_{i(m)} = \alpha_{i(m)}]$. ∎

This lemma implies that it suffices to prove theorem 3.13 only for energies of two variables.

Let $\bar{E}(x_1, x_2)$ be a graph-representable function of two variables. Let us prove that $\bar{E}$ is regular, i.e. that $A \leq 0$ where $A = \pi(\bar{E}) = \bar{E}(0,0) + \bar{E}(1,1) - \bar{E}(0,1) - \bar{E}(1,0)$. Suppose this is not true: $A > 0$.

We can write

$$\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & A \\ \hline\end{array} = \bar{E}(0,0) + \begin{array}{|c|c|}\hline 0 & -\bar{E}(0,1) \\ \hline 0 & -\bar{E}(0,1) \\ \hline\end{array} + \begin{array}{|c|c|}\hline 0 & 0 \\ \hline -\bar{E}(1,0) & -\bar{E}(1,0) \\ \hline\end{array} + \begin{array}{|c|c|}\hline \bar{E}(0,0) & \bar{E}(0,1) \\ \hline \bar{E}(1,0) & \bar{E}(1,1) \\ \hline\end{array}$$

All functions on the right side are graph-representable, therefore by the additivity theorem (see section 3.8) the function E is graph-representable as well, where

$$E = \begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & A \\ \hline\end{array}$$

Consider a graph $\mathcal{G}$ and a set $\mathcal{V}_0 = \{v_1, v_2\}$ representing E. It means that there is a constant K such that $\mathcal{G}$, $\mathcal{V}_0$ exactly represent $E'(x_1, x_2) = E(x_1, x_2) + K$:

$$E' = \begin{array}{|c|c|}\hline K & K \\ \hline K & K + A \\ \hline\end{array}$$

The cost of a minimum s-t-cut on $\mathcal{G}$ is K (since this cost is just the minimum entry in the table for E'); hence, $K \geq 0$. Thus the value of the maximum flow from s to t in $\mathcal{G}$ is K. Let $\mathcal{G}^0$ be the residual graph obtained from $\mathcal{G}$ after pushing the flow K. Let $E^0(x_1, x_2)$ be the function exactly represented by $\mathcal{G}^0, \mathcal{V}_0$.

By the definition of graph representability, $E'(\alpha_1, \alpha_2)$ is equal to the value of a minimum cut (or maximum flow) on the graph $\mathcal{G}[x_1 = \alpha_1, x_2 = \alpha_2]$. The following sequence of operations shows one possible way to push a maximum flow through this graph.

- First we take the original graph $\mathcal{G}$ and push the flow K; then we get the residual graph $\mathcal{G}^0$. (It is equivalent to pushing flow through $\mathcal{G}[x_1 = \alpha_1, x_2 = \alpha_2]$ where we do not use edges corresponding to constraints $x_1 = \alpha_1$ and $x_2 = \alpha_2$).

- Then we add edges corresponding to these constraints; then we get the graph $\mathcal{G}^0[x_1 = \alpha_1, x_2 = \alpha_2]$.

- Finally we push the maximum flow possible through the graph $\mathcal{G}^0[x_1 = \alpha_1, x_2 = \alpha_2]$; the amount of this flow is $E^0(\alpha_1, \alpha_2)$ according to the definition of graph representability.

The total amount of flow pushed during all steps is $K + E^0(\alpha_1, \alpha_2)$; thus,

$$E'(\alpha_1, \alpha_2) = K + E^0(\alpha_1, \alpha_2)$$

or

$$E(\alpha_1, \alpha_2) = E^0(\alpha_1, \alpha_2)$$

We proved that E is exactly represented by $\mathcal{G}^0$, $\mathcal{V}_0$.

The value of a minimum cut/maximum flow on $\mathcal{G}^0$ is 0 (it is the minimum entry in the table for E); thus, there is no augmenting path from s to t in $\mathcal{G}^0$. However, if we add edges (v_1, t) and (v_2, t) then there will be an augmenting path from s to t in $\mathcal{G}^0[x_1 = \alpha_1, x_2 = \alpha_2]$ since $E(1, 1) = A > 0$. Hence, this augmenting path will contain at least one of these edges and, therefore, either v_1 or v_2 will be in the path. Let P be the part of this path going from the source until v_1 or v_2 is first encountered. Without loss of generality we can assume that it will be v_1. Thus, P is an augmenting path from s to v_1 which does not contain edges that we added, namely (v_1, t) and (v_2, t).

Finally let us consider the graph $\mathcal{G}^0[x_1 = 1, x_2 = 0]$ which is obtained from $\mathcal{G}^0$ by adding edges (v_1, t) and (s, v_2) with infinite capacities. There is an augmenting path $\{P, (v_1, t)\}$ from the source to the sink in this graph; hence, the value of a minimum cut/maximum flow on it greater than zero, or $E(1, 0) > 0$. We get a contradiction.

3.6 Related work

There is an interesting relationship between regular functions and submodular functions.[3] Let $\mathcal{U}$ be a finite set and $g : 2^{\mathcal{U}} \to \mathcal{R}$ be a real-valued function defined on the set of all subsets of $\mathcal{U}$. g is called *submodular* if for any $X, Y \subset \mathcal{U}$

$$g(X) + g(Y) \geq g(X \cup Y) + g(X \cap Y).$$

See [Fuj90], for example, for a discussion of submodular functions. An equivalent definition of submodular functions is that g is called submodular if for any $X \subset \mathcal{U}$ and $i, j \in \mathcal{U} - X$

$$g(X \cup \{j\}) - g(X) \geq g(X \cup \{i, j\}) - g(X \cup \{i\}).$$

Obviously, functions of subsets X of $\mathcal{U} = \{1, \ldots, n\}$ can be viewed as functions of n binary variables $(x_1, \ldots, x_n)$; the indicator variable x_i is 1 if i is included in X and 0 otherwise. Then it is easy to see that the second definition of submodularity reduces to the definition of regularity. Thus, submodular functions and regular functions are essentially the same. We use different names to emphasize a technical distinction between them: submodular functions are functions of subsets of $\mathcal{U}$ while regular functions are functions of binary variables. From our experience, the second point of view is much more convenient for vision applications.

Submodular functions have received a lot of attention in combinatorial optimization literature. A remarkable fact about submodular functions is that they can be minimized in polynomial time ([IFF00],[Sch00]). A function g is assumed to be given as a *value oracle*, i.e., a "black box" which, for any input subset X returns the value $g(X)$.

[3] We thank Yuri Boykov and Howard Karloff for pointing out this connection.

Unfortunately, algorithms for minimizing arbitrary submodular functions are extremely slow. For example, the algorithm of [IFF00] runs in $O(n^5 \min\{\log nM, n^2 \log n\})$ time where M is an upper bound on $|g(X)|$ among $X \in 2^{\mathcal{U}}$. Thus, our work can be viewed as identifying an important subclass of submodular functions for which much faster algorithm can be used.

In addition, a relation between submodular functions and certain graph-representable functions called *cut functions* is already known. Using our terminology, cut functions can be defined as functions which can be represented by a graph without the source and the sink and without auxiliary vertices. It is well-known that cut functions are submodular. [Cun85] characterizes cut functions and gives a general-purpose graph construction for them.

It can be shown that the set of cut functions is a strict subset of $\mathcal{F}^2$. We allow a more general graph construction; as a result, we can minimize a larger class of functions, namely regular functions in $\mathcal{F}^3$.

3.7 Summary of graph constructions

We now summarize the graph constructions used for regular functions. Recall that for a function $E(x_1, \ldots, x_n)$ we define

$$\pi(E) = \sum_{x_1 \in \{0,1\}, \ldots, x_n \in \{0,1\}} (\Pi_{i=1}^{n} (-1)^{x_i})\, E(x_1, \ldots, x_n).$$

The notation *edge*(v, c) will mean that we add an edge (s, v) with the weight c if $c > 0$, or an edge (v, t) with the weight $-c$ if $c < 0$.

3.7.1 Regular functions of one binary variable

Recall that all functions of one variable are regular. For a function $E(x_1)$, we construct a graph $\mathcal{G}$ with three vertices $\mathcal{V} = \{v_1, s, t\}$. There is a single edge $edge(v_1, E(1) - E(0))$.

3.7.2 Regular functions of two binary variables

We now show how to construct a graph $\mathcal{G}$ for a regular function $E(x_1, x_2)$ of two variables. It will contain four vertices: $\mathcal{V} = \{v_1, v_2, s, t\}$. The edges $\mathcal{E}$ are given below.

- $edge(v_1, E(1,0) - E(0,0))$;
- $edge(v_2, E(1,1) - E(1,0))$;
- (v_1, v_2) with the weight $-\pi(E)$.

3.7.3 Regular functions of three binary variables

We next show how to construct a graph $\mathcal{G}$ for a regular function $E(x_1, x_2, x_3)$ of three variables. It will contain five vertices: $\mathcal{V} = \{v_1, v_2, v_3, u, s, t\}$. If $\pi(E) \geq 0$ then the edges are

- $edge(v_1, E(1,0,1) - E(0,0,1))$;
- $edge(v_2, E(1,1,0) - E(1,0,0))$;
- $edge(v_3, E(0,1,1) - E(0,1,0))$;
- (v_2, v_3) with the weight $-\pi(E[x_1 = 0])$;
- (v_3, v_1) with the weight $-\pi(E[x_2 = 0])$;

- (v_1, v_2) with the weight $-\pi(E[x_3 = 0])$;
- (v_1, u), (v_2, u), (v_3, u), (u, t) with the weight $\pi(E)$.

If $\pi(E) < 0$ then the edges are

- $edge(v_1, E(1, 1, 0) - E(0, 1, 0))$;
- $edge(v_2, E(0, 1, 1) - E(0, 0, 1))$;
- $edge(v_3, E(1, 0, 1) - E(1, 0, 0))$;
- (v_3, v_2) with the weight $-\pi(E[x_1 = 1])$;
- (v_1, v_3) with the weight $-\pi(E[x_2 = 1])$;
- (v_2, v_1) with the weight $-\pi(E[x_3 = 1])$;
- (u, v_1), (u, v_2), (u, v_3), (s, u) with the weight $-\pi(E)$.

3.8 Technical details

In this section, we provide a few additional technical details that were deferred earlier in this chapter.

3.8.1 Proof of the NP-hardness theorem

We now give a proof of the NP-hardness theorem (theorem 3.4), which shows that in the absence of regularity it is intractable to minimize even energy functions in $\mathcal{F}^2$.

PROOF: Adding functions of one variable does not change the class of functions that we are considering. Thus, we can assume without loss of generality that E^2

has the form

$$E^2 = \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & A \\ \hline \end{array}$$

(We can transform an arbitrary function of two variables to this form as follows: we subtract a constant from the first row to make the upper left element zero, then we subtract a constant from the second row to make the bottom left element zero, and finally we subtract a constant from the second column to make the upper right element zero.)

These transformations preserve the functional π, so E^2 is non-regular, which means that $A > 0$.

We will prove the theorem by reducing the maximum independent set problem, which is known to be NP-hard, to our energy minimization problem.

Let an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the input to the maximum independent set problem. A subset $\mathcal{U} \subset \mathcal{V}$ is said to be independent if for any two vertices $u, v \in \mathcal{U}$ the edge (u, v) is not in $\mathcal{E}$. The goal is to find an independent subset $\mathcal{U}^* \subset \mathcal{V}$ of maximum cardinality. We construct an instance of the energy minimization problem as follows. There will be $n = |\mathcal{V}|$ binary variables $x_1, \ldots, x_n$ corresponding to the vertices $v_1, \ldots, v_n$ of $\mathcal{V}$. Let us consider the energy

$$E(x_1, \ldots, x_n) = \sum_i E^i(x_i) + \sum_{(i,j) \in \mathcal{N}} E^2(x_i, x_j)$$

where $E^i(x_i) = -\frac{A}{2n} \cdot x_i$ and $\mathcal{N} = \{(i,j) \mid (v_i, v_j) \in \mathcal{E}\}$.

There is a one-to-one correspondence between all configurations $(x_1, \ldots, x_n)$ and subsets $\mathcal{U} \subset \mathcal{V}$: a vertex v_i is in $\mathcal{U}$ if and only if $x_i = 1$. Moreover, the first term of $E(x_1, \ldots, x_n)$ is $-\frac{A}{2n}$ times the cardinality of U (which cannot be less than $-\frac{A}{2}$), and the second term is 0, if U is independent, and at least A otherwise. Thus, the minimum of the energy yields the independent subset of maximum cardinality.

3.8.2 The additivity theorem

Theorem (additivity) The sum of two graph-representable functions is graph-representable.

It is important to note that the proof of this theorem is constructive. The construction has a particularly simple form if the graphs representing the two functions are defined on the same set of vertices (i.e., they differ only in their edge weights). In this case, by simply adding the edge weights together, we obtain a graph that represents the sum of the two functions. If one of the graphs has no edge between two vertices, we can add an edge with weight 0.

PROOF:Let us assume for simplicity of notation that E' and E'' are functions of all n variables: $E' = E'(x_1, \ldots, x_n)$, $E'' = E''(x_1, \ldots, x_n)$. By the definition of graph representability, there exist constants K', K'', graphs $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, $\mathcal{G}'' = (\mathcal{V}'', \mathcal{E}'')$ and the set $\mathcal{V}_0 = \{v_1, \ldots, v_n\}$, $\mathcal{V}_0 \subset \mathcal{V}' - \{s, t\}$, $\mathcal{V}_0 \subset \mathcal{V}'' - \{s, t\}$ such that $E' + K'$ is exactly represented by $\mathcal{G}'$, $\mathcal{V}_0$ and $E'' + K''$ is exactly represented by $\mathcal{G}''$, $\mathcal{V}_0$. We can assume that the only common vertices of $\mathcal{G}'$ and $\mathcal{G}''$ are $\mathcal{V}_0 \cup \{s, t\}$. Let us construct the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as the combined graph of $\mathcal{G}'$ and $\mathcal{G}''$: $\mathcal{V} = \mathcal{V}' \cup \mathcal{V}''$, $\mathcal{E} = \mathcal{E}' \cup \mathcal{E}''$.

Let $\tilde{E}$ be the function that $\mathcal{G}$, $\mathcal{V}_0$ exactly represents. Let us prove that $\tilde{E} \equiv E + (K' + K'')$ (and, therefore, E is graph-representable).

Consider a configuration $x_1, \ldots, x_n$. Let $C' = cut(\mathcal{S}', \mathcal{T}')$ be the cut on $\mathcal{G}'$ with the smallest cost among all cuts for which $v_i \in \mathcal{S}'$ if $x_i = 0$, and $v_i \in \mathcal{T}'$ if $x_i = 1$ $(1 \leq i \leq n)$. According to the definition of graph representability,

$$E'(x_1, \ldots, x_n) + K' = \sum_{u \in \mathcal{S}', v \in \mathcal{T}', (u,v) \in \mathcal{E}'} c(u, v)$$

Let $C'' = \mathcal{S}'', \mathcal{T}''$ be the cut on $\mathcal{G}''$ with the smallest cost among all cuts for which

$v_i \in \mathcal{S}''$ if $x_i = 0$, and $v_i \in \mathcal{T}''$ if $x_i = 1$ $(1 \leq i \leq n)$. Similarly,

$$E''(x_1, \ldots, x_n) + K'' = \sum_{u \in \mathcal{S}'', v \in \mathcal{T}'', (u,v) \in \mathcal{E}''} c(u, v)$$

Let $\mathcal{S} = \mathcal{S}' \cup \mathcal{S}''$, $\mathcal{T} = \mathcal{T}' \cup \mathcal{T}''$. It is easy to check that $C = \mathcal{S}, \mathcal{T}$ is a cut on $\mathcal{G}$. Thus,

$$\tilde{E}(x_1, \ldots, x_n) \leq \sum_{u \in \mathcal{S}, v \in \mathcal{T}, (u,v) \in \mathcal{E}} c(u, v)$$

$$= \sum_{u \in \mathcal{S}', v \in \mathcal{T}', (u,v) \in \mathcal{E}'} c(u, v) + \sum_{u \in \mathcal{S}'', v \in \mathcal{T}'', (u,v) \in \mathcal{E}''} c(u, v)$$

$$= (E'(x_1, \ldots, x_n) + K') + (E''(x_1, \ldots, x_n) + K'') = E(x_1, \ldots, x_n) + (K' + K'').$$

Now let $C = \mathcal{S}, \mathcal{T}$ be the cut on $\mathcal{G}$ with the smallest cost among all cuts for which $v_i \in \mathcal{S}$ if $x_i = 0$, and $v_i \in \mathcal{T}$ if $x_i = 1$ $(1 \leq i \leq n)$, and let $\mathcal{S}' = \mathcal{S} \cap \mathcal{V}'$, $\mathcal{T}' = \mathcal{T} \cap \mathcal{V}'$, $\mathcal{S}'' = \mathcal{S} \cap \mathcal{V}''$, $\mathcal{T}'' = \mathcal{T} \cap \mathcal{V}'$. It is easy to see that $C' = \mathcal{S}', \mathcal{T}'$ and $C'' = \mathcal{S}'', \mathcal{T}''$ are cuts on $\mathcal{G}'$ and $\mathcal{G}''$, respectively. According to the definition of graph representability,

$$E(x_1, \ldots, x_n) + (K' + K'') = (E'(x_1, \ldots, x_n) + K') + (E''(x_1, \ldots, x_n) + K'')$$

$$\leq \sum_{u \in \mathcal{S}', v \in \mathcal{T}', (u,v) \in \mathcal{E}'} c(u, v) + \sum_{u \in \mathcal{S}'', v \in \mathcal{T}'', (u,v) \in \mathcal{E}''} c(u, v)$$

$$= \sum_{u \in \mathcal{S}, v \in \mathcal{T}, (u,v) \in \mathcal{E}'} c(u, v) = \tilde{E}(x_1, \ldots, x_n).$$

■

3.8.3 Properties of the functional π

While the functional π has a form that at first seems counterintuitive, it actually has a close relationship with the classes of energy functions we are concerned with. To see this, let us define the class $\mathcal{F}^k$ to be functions of binary variables that can be written as a sum of functions of up to k variables at a time,

$$E(x_1, \ldots, x_n) = \sum_{i(1)<\ldots<i(k'),k'\leq k} E^{i(1),\ldots,i(k')}(x_{i(1)}, \ldots, x_{i(k')}) \tag{3.6}$$

(it is an obvious generalization of classes $\mathcal{F}^2$ and $\mathcal{F}^3$).

In this section we will show that the functional π can serve as an indicator of the class of a function.

Theorem 3.17 *Suppose, E is a function of n binary variables ($n \geq 1$). Then $E \in \mathcal{F}^{n-1}$ if and only if $\pi(E) = 0$.*

PROOF: One direction of the theorem is trivial: if $E \in \mathcal{F}^{n-1}$, then $\pi(E) = 0$ by lemma 3.10. Let us prove $\pi(E) = 0$ implies $E \in \mathcal{F}^{n-1}$ by induction on n.

Induction base: $n = 1$ (E is a function of one variable). If $\pi(E) = E(0)-E(1) = 0$, then $E(0) = E(1)$, i.e. E is constant and, therefore, is in $\mathcal{F}^0$.

Now suppose that the theorem is true for $n \geq 1$. Let us consider a function $E(x_1, \ldots, x_{n+1})$ of $n+1$ binary variables such that $\pi(E) = 0$.

Let $E_0 = E[x_{n+1} = 0]$, $E_1 = E[x_{n+1} = 1]$ be functions of n variables $x_1, \ldots, x_n$. It is easy to check that $\pi(E) = \pi(E_0) - \pi(E_1)$. Thus, $\pi(E_0) = \pi(E_1) = C$.

Let $R[C]$ be a function of n variables $x_1, \ldots, x_n$ such that $\pi(R[C]) = C$. (For example, we can define it as $R[C](x_1, \ldots, x_n) = C$, if $x_1 = 0, \ldots, x_n = 0$, and $R[C](x_1, \ldots, x_n) = 0$ otherwise.)

Let

$$E'(x_1, \ldots, x_{n+1}) = E(x_1, \ldots, x_{n+1}) - R[C](x_1, \ldots, x_n)$$

$$E'_0(x_1, \ldots, x_n) = E_0(x_1, \ldots, x_n) - R[C](x_1, \ldots, x_n)$$

$$E'_1(x_1, \ldots, x_n) = E_1(x_1, \ldots, x_n) - R[C](x_1, \ldots, x_n)$$

E' is a function of $n+1$ variables, E'_0, E'_1 are functions of n variables. Clearly, $E'_0 = E'[x_{n+1} = 0]$, $E'_1 = E'[x_{n+1} = 1]$, $\pi(E'_1) = \pi(E'_0) = \pi(E_0) - \pi(R[C]) = C - C = 0$.

By the induction hypothesis, $E'_0, E'_1 \in \mathcal{F}^{n-1}$:

$$E'_0(x_1, \dots, x_n) = \sum_\alpha E_0^\alpha(x_1, \dots, x_n)$$
$$E'_1(x_1, \dots, x_n) = \sum_\beta E_1^\beta(x_1, \dots, x_n)$$

where all terms E_0^α, E_1^β depend on at most $n-1$ variables.

Let $\tilde{E}_0^\alpha$, $\tilde{E}_1^\beta$, $\tilde{E}$ be functions of $n+1$ variables defined as follows:

$$\tilde{E}_0^\alpha(x_1, \dots, x_{n+1}) = \begin{cases} E_0^\alpha(x_1, \dots, x_n), & x_{n+1} = 0 \\ 0, & x_{n+1} = 1 \end{cases}$$
$$\tilde{E}_1^\beta(x_1, \dots, x_{n+1}) = \begin{cases} 0, & x_{n+1} = 0 \\ E_1^\beta(x_1, \dots, x_n), & x_{n+1} = 1 \end{cases}$$
$$\tilde{E} \equiv \sum_\alpha \tilde{E}_0^\alpha + \sum_\beta \tilde{E}_1^\beta$$

We can write

$$\tilde{E}[x_{n+1} = 0] \equiv \sum_\alpha \tilde{E}_0^\alpha[x_{n+1} = 0] + \sum_\beta \tilde{E}_1^\beta[x_{n+1} = 0] \equiv \sum_\alpha E_0^\alpha + 0 \equiv E'[x_{n+1} = 0]$$
$$\tilde{E}[x_{n+1} = 1] \equiv \sum_\alpha \tilde{E}_0^\alpha[x_{n+1} = 1] + \sum_\beta \tilde{E}_1^\beta[x_{n+1} = 1] \equiv 0 + \sum_\alpha E_1^\beta \equiv E'[x_{n+1} = 1]$$

Thus, $E' \equiv \tilde{E}$. All terms $\tilde{E}_0^\alpha$, $\tilde{E}_1^\beta$ depend on at most n variables, therefore, $E' \in \mathcal{F}^n$ and $E \in \mathcal{F}^n$ (since E is a sum of E' and $R[C]$, and $R[C]$ depends on n variables).

Theorem 3.18 *Suppose, E is a function of m binary variables ($m \geq n \geq 1$). Then $E \in \mathcal{F}^{n-1}$ if and only if $\pi(E') = 0$ for all functions E' that are projections of E of at least n variables (i.e. at most $m-n$ variables are fixed).*

PROOF: One direction of the theorem is trivial: if $E \in \mathcal{F}^{n-1}$, then all projections E' of E are also in $\mathcal{F}^{n-1}$, therefore $\pi(E') = 0$ by lemma 3.10. Let us prove the opposite direction by induction on m.

The induction base (the case $m = n$) follows from theorem 3.17.

Now suppose that it is true for $m \geq n$. Let us consider a function $E(x_1, \ldots, x_{m+1})$ of $m+1$ binary variables such that $\pi(E') = 0$ for all projections E' of E of at least n binary variables.

Any projection of $E[x_{m+1} = 0]$ is also a projection of E. Thus, $\pi(E') = 0$ for all projections E' of $E[x_{m+1} = 0]$ of at least n binary variables. $E[x_{m+1} = 0]$ is a function of m variables; by the induction hypothesis, $E[x_{m+1} = 0] \in \mathcal{F}^{n-1}$. Similarly, $E[x_{m+1} = 1] \in \mathcal{F}^{n-1}$.

Let E_0, E_1 be functions of $m+1$ variables $x_1, \ldots, x_{m+1}$ defined as follows:

$$E_0(x_1, \ldots, x_{m+1}) = \begin{cases} E[x_{m+1} = 0](x_1, \ldots, x_m), & x_{m+1} = 0 \\ 0, & x_{m+1} = 1 \end{cases}$$

$$E_1(x_1, \ldots, x_{m+1}) = \begin{cases} 0, & x_{m+1} = 0 \\ E[x_{m+1} = 1](x_1, \ldots, x_m), & x_{m+1} = 1 \end{cases}$$

It is easy to see that E_0 and E_1 are in $\mathcal{F}^n$ since $E[x_{m+1} = 0]$ and $E[x_{m+1} = 1]$ are in $\mathcal{F}^{n-1}$. Therefore, $E \equiv E_0 + E_1$ is in $\mathcal{F}^n$ as well.

Thus, E can be written as

$$E(x_1, \ldots, x_{m+1}) = \sum_{i(1) < \ldots < i(n)} \tilde{E}^{i(1), \ldots, i(n)}(x_{i(1)}, \ldots, x_{i(n)}) =$$

$$= \sum_{i(1) < \ldots < i(n)} E^{i(1), \ldots, i(n)}(x_1, \ldots, x_{m+1})$$

where

$$E^{i(1), \ldots, i(n)}(x_1, \ldots, x_{m+1}) = \tilde{E}^{i(1), \ldots, i(n)}(x_{i(1)}, \ldots, x_{i(n)})$$

Let us a take a projection of E of n variables where the variables that are not fixed are $x_{i(1)}, \ldots, x_{i(n)}$ ($i(1) < \ldots < i(n)$), and apply the functional π to this projection. Let us denote this operation as π'.

Let us consider the result of this operation on a single term $E^{i'(1),\ldots,i'(n)}$, $i'(1) < \ldots < i'(n)$. If $(i'(1), \ldots, i'(n))$ coincides with $(i(1), \ldots, i(n))$ then $\pi'(E^{i'(1),\ldots,i'(n)}) = \pi(\tilde{E}^{i'(1),\ldots,i'(n)})$. Suppose that these indexes do not coincide, then $i(k)$ is not in $\{i'(1), \ldots, i'(n)\}$ for some k. $E^{i'(1),\ldots,i'(n)}$ does not depend on $x_{i(k)}$. Therefore, any projection of this function where $x_{i(k)}$ is not fixed also does not depend on $x_{i(k)}$, so the functional π of such projection is 0 by lemma 3.10. Thus, $\pi'(E^{i'(1),\ldots,i'(n)}) = 0$.

Therefore, $\pi'(E) = \pi'(E^{i(1),\ldots,i(n)}) = \pi(\tilde{E}^{i(1),\ldots,i(n)})$. By assumption $\pi'(E) = 0$, so $\tilde{E}^{i(1),\ldots,i(n)} \in \mathcal{F}^{n-1}$ by theorem 3.17.

E is a sum of terms which lie in $\mathcal{F}^{n-1}$, therefore E is in $\mathcal{F}^{n-1}$ as well.

We can also show that π is unique (up to a constant multiplicative factor) among the linear functionals that are indicator variables for the class of an energy function.

Theorem 3.19 *Suppose, π' is a linear functional mapping the set of functions E of n binary variables to the set of real numbers with the property that $E \in \mathcal{F}^{n-1}$ if and only if $\pi'(E) = 0$. Then there exists a constant $c \neq 0$ such that $\pi'(E) = c\pi(E)$ for any function E.*

PROOF: π' can be written as

$$\pi'(E) = \sum_{x_1 \in \{0,1\}, \ldots, x_n \in \{0,1\}} c(x_1, \ldots, x_n) \cdot E(x_1, \ldots, x_n)$$

where $c(x_1, \ldots, x_n)$ are some constants.

Let $c = c(0, \ldots, 0)$. Let us prove that $c(x_1, \ldots, x_n) = (\Pi_{i=1}^{n} (-1)^{x_i})\, c$ by induction on $d = x_1 + \ldots + x_n$.

Induction base: if $d = x_1 + \ldots + x_n = 0$, then $x_1 = \ldots = x_n = 0$, so $c(x_1, \ldots, x_n) = c$.

Suppose that it is true for all configurations $x_1, \ldots, x_n$ such that $x_1 + \ldots + x_n = d$ ($d \geq 0$). Let us consider a configuration $x_1, \ldots, x_n$ such that $x_1 + \ldots + x_n = d + 1$. At least one of the variables is 1 since $d + 1 \geq 1$. Let us assume that this is the first variable (we can ensure it by renaming indexes): $x_1 = 1$. Let us consider the function $E(x'_1, \ldots, x'_n)$ which is 1 if $(x'_2, \ldots, x'_n) = (x_2, \ldots, x_n)$, and 0 otherwise. Clearly,

$$\pi'(E) = c(0, x_2, \ldots, x_n) + c(1, x_2, \ldots, x_n)$$

$0 + x_2 + \ldots + x_n = d$, therefore $c(0, x_2, \ldots, x_n) = (\Pi_{i=2}^{n} (-1)^{x_i})\, c$ by the induction hypothesis. E is in $\mathcal{F}^{n-1}$ since it does not depend on x_1. Thus, $\pi'(E)$ must be 0, so

$$c(1, x_2, \ldots, x_n) = -c(0, x_2, \ldots, x_n) = (\Pi_{i=1}^{n} (-1)^{x_i})\, c$$

We proved that $\pi'(E) = c\pi(E)$ for any function E. c cannot be 0 since otherwise $\pi'(E)$ would always be 0, and there are functions E of n variables which are not in $\mathcal{F}^{n-1}$.

Chapter 4

Multi-camera scene reconstruction

In this chapter we present several algorithms for scene reconstruction from multiple viewpoints. The algorithms are based on an energy minimization formulation. They have many important properties which are essential for obtaining accurate results. These properties are listed below. Our algorithms have all of these properties.

Ideally, an algorithm for multi-camera scene reconstruction should

- Work with an arbitrary number of cameras,
- Enforce the photoconsistency constraint,
- Enforce the visibility constraint,
- Impose spatial smoothness while preserving discontinuities,
- Treat all input images symmetrically, and

In addition, of course, the algorithm should run efficiently and give strong experimental results. This is most naturally obtained by energy minimization using graph cuts.

We now discuss each property briefly.

Number of cameras

Traditionally, researchers have focused their attention on the two-camera stereo problem. This can be formulated as a visual correspondence problem: we want to find a mapping from one image to the other satisfying the uniqueness constraint. Thus, in principle it can be solved without referring to 3D geometry of the scene. Unfortunately, this approach is not readily generalizable to multiple cameras. More complex scene representations are needed which take into account 3D geometry.

Photoconsistency constraint

The constraint says that if two pixels in different cameras are looking at the same element of the scene, then their intensities should be similar (this is often called the Lambertian, or constant brightness, assumption). Most stereo algorithms enforce this constraint, however there are several exceptions. For example, algorithms based on silhouette intersections use only information obtained by subtracting images with an object and images without an object ("background" images).

Visibility constraint

This is a generalization to multiple cameras of the uniqueness constraint used in two-camera stereo. Similar to the two-camera case, this constraint is particularly important for obtaining accurate results at object boundaries and for detecting occlusions. The visibility constraint is described in more detail later the chapter. Unlike other constraints, this is a hard constraint, i.e. the penalty for its violation is infinity. It imposes complex nonlinear restrictions on the search space.

Spatial smoothness

A spatially smooth solution should be preferred, in situations where the input data is ambiguous. Unfortunately, since vision is concerned with extracting information about the 3D scene from one or more 2D images, ambiguity is inevitable. Moreover, it is important not to over-smooth the output, as discussed in section 2.3.

Symmetry

Many stereo algorithms do not treat all input images equally, and have a preferred image instead. They will give different results if a different preferred image is chosen. However, there is no asymmetry in the formulation of the scene reconstruction problem. This is an indication of the fact that these algorithms are not well-suited to this problem.

Energy minimization

Our energy function is not the first that captures all constraints mentioned above; see [KSC01], for example. However, the minimization algorithm in [KSC01] is an iteration of two steps one of which is a local optimization. It can easily be trapped in local minima. As a result, this approach produces poor results.

In contrast, our minimization technique is based on the global α-expansion operation described in section 2.4. The output of our algorithm has the same strong property as the expansion move algorithm of [BVZ01]: no α-expansion can decrease the energy.

The algorithms described in this chapter are based on papers [KZ01, KZ02a, KZG03]. In the first two papers we presented two different stereo algorithms based on graph cuts (the first one for two cameras, and the second one for multiple cameras). The relationship between these two approaches was not clear at the time. In the third paper we presented a unified algorithm and showed that algorithms in [KZ01, KZ02a] are its special cases. The energy function for the unified algorithm contains two different smoothness terms. If one of them is omitted, then the algorithm reduces to [KZ01] or [KZ02a], respectively.

The rest of the chapter is organized as follows. In section 4.1 we review other algorithms for two-camera stereo and multiple-camera stereo. In section 4.2 we formalize our problem and present the energy function for the unified algorithm. Two special cases are considered in section 4.3. Minimization of the energy function is discussed in section 4.4. Experimental results are presented in section 4.5. Conclusions are given in section 4.6.

4.1 Related work

The problem of reconstructing a scene from multiple cameras has received a great deal of attention in the last few years. One extensively-explored approach to this problem is voxel occupancy. In voxel occupancy [MA83, Sze93] the scene is represented as a set of 3-dimensional voxels, and the task is to label the individual voxels as filled or empty. Voxel occupancy is typically solved using silhouette intersection, usually from multiple cameras but sometimes from a single camera with the object placed on a turntable [CB92]. It is known that the output of silhouette intersection even without noise is not the actual 3-dimensional shape, but rather an approximation called the visual hull [Lau94].

4.1.1 Voxel coloring and space carving

Voxel occupancy, however, fails to exploit the consistent appearance of a scene element between different cameras. This constraint, called *photo-consistency*, is obviously quite powerful. Two well-known recent algorithms that have used photo-consistency are voxel coloring [SD99] and space carving [KS00].

Voxel coloring makes a single pass through voxel space, first computing the visibility of each voxel and then its color. There is a constraint on the camera geometry, namely that no scene point is allowed to be within the convex hull of the camera centers. As we will see, our approach handles all the camera configurations where voxel coloring can be used. Space carving is another voxel-oriented approach that uses the photo-consistency constraint to prune away empty voxels from the volume. Space carving has the advantage of allowing arbitrary camera geometry.

One major limitation of voxel coloring and space carving is that they lack a way of imposing spatial coherence. This is particularly problematic because the image data is almost always ambiguous. Another (related) limitation comes from the fact that these methods traverse the volume making "hard" decisions concerning the occupancy of each voxel they analyze. Because the data is ambiguous, such a decision can easily be incorrect, and there is no easy way to undo such a decision later on.

4.1.2 Energy minimization approaches

Although [RC98] and [SVZ00] use energy minimization via graph cuts, their focus is quite different from ours. [RC98] uses an energy function whose global minimum can be computed efficiently via graph cuts; however, the spatial smoothness term is not discontinuity preserving, and so the results tend to be oversmoothed. Visibility constraints are not used. [SVZ00] computes the global minimum of a different energy function as an alternative to silhouette intersection (i.e., to determine voxel occupancy). Their approach does not deal with photoconsistency at all, nor do they reason about visibility.

Our method is also related to the work of [KSC01], which also relies on graph cuts. They extend the work of [BVZ01], which focused on traditional stereo matching, to allow an explicit label for occluded pixels. While the energy function that they use is of a similar general form to ours, they do not treat the input images symmetrically. While we effectively compute a disparity map with respect to each camera, they compute a disparity map only with respect to a single camera.

4.2 Problem formulation

Now we will formalize the problem we are trying to solve. We will introduce two mappings describing the geometry of the scene, and enforce hard constraints between them. These mappings will be similar to the ones used in [KZ02a] and [KZ01], respectively.

Suppose we are given n calibrated images of the same scene taken from different viewpoints (or at different moments of time). Let $\mathcal{P}_i$ be the set of pixels in the camera i, and let $\mathcal{P} = \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n$ be the set of all pixels. A pixel $p \in \mathcal{P}$ corresponds to a ray in 3D-space. Our first mapping f will describe depths for all pixels. More formally, the labeling f is a mapping from $\mathcal{P}$ to $\mathcal{L}$ where $\mathcal{L}$ is a discrete set of labels corresponding to different depths. In the current implementation of our method, labels correspond to increasing depth from a fixed camera.

A pair $\langle p, l \rangle$ where $p \in \mathcal{P}$, $l \in \mathcal{L}$ corresponds to some point in 3D-space. We will refer to such pairs as *3D-points*.

Our method has the same limitation as voxel coloring [SD99]. Namely, there must exist a function $\mathcal{D} : R^3 \mapsto R$ such that for all scene points P and Q, P occludes Q in a camera i only if $\mathcal{D}(P) < \mathcal{D}(Q)$. If such a function exists then labels correspond to level sets of this function. In our current implementation, we make a slightly more specific assumption, which is that the cameras must lie in one semiplane looking at the other semiplane. The interpretation of labels will be as follows: each label corresponds to a plane in 3D-space, and a 3D-point $\langle p, l \rangle$ is the intersection of the ray corresponding to the pixel p and the plane l.

Let us introduce the set of interactions I consisting of (unordered) pairs of 3D-points $\langle p_1, l_1 \rangle$, $\langle p_2, l_2 \rangle$ "close" to each other in 3D-space. We will discuss several criteria for "closeness" later. In general, I can be an arbitrary set of pairs of 3D-points satisfying the following constraint:

- Only 3D-points at the same depth can interact, i.e.if $\{\langle p_1, l_1 \rangle, \langle p_2, l_2 \rangle\} \in I$ then $l_1 = l_2$.

To simplify the notation, we will denote interactions in I as $\langle p, q, l \rangle$ where p, q are pixels and l is a depth label.

Since 3D-points $\langle p, l \rangle$ and $\langle q, l \rangle$ are close to each other, the interaction $\langle p, q, l \rangle$ approximately corresponds to a single point in 3D-space (it can be, for example, the middle point between $\langle p, l \rangle$ and $\langle q, l \rangle$). We can describe the geometry of the scene by specifying which interactions are visible. Let us introduce another mapping $g : I \to \{0, 1\}$. $g(\langle p, q, l \rangle)$ will be 1 if this interaction is visible in both pixels p and q, and 0 otherwise. This mapping allows us to introduce the data term (i.e. the photoconsistency constraint) very naturally: we will enforce photoconsistency between p and q only if the interaction $\langle p, q, l \rangle$ is *active*, i.e. $g(\langle p, q, l \rangle) = 1$.

4.2.1 Our energy function

Now we will define the energy function that we minimize. It will consist of five terms:

$$E(f, g) = E_{data}(g) + E^{(1)}_{smooth}(f) + E^{(2)}_{smooth}(g) + E_{vis}(f) + E_{consistency}(f, g) \quad (4.1)$$

The terms $E^{(1)}_{smooth}$ and E_{vis} were used in [KZ02a]. The terms $E^{(2)}_{smooth}$ and E_{data} are similar to the ones used in [KZ01]. The term $E_{consistency}$ will enforce consistency between the mappings f and g.

Data term

The data term will be

$$E_{data}(g) = \sum_{i \in I} D_i(g(i))$$

where $D_i(0) = K$ for some constant K and $D_i(1)$ depends on intensities of pixels p and q involved in the interaction i. We can have, for example, $D_{\langle p,q,l\rangle}(1) = (Intensity(p) - Intensity(q))^2$.

Smoothness terms

Two smoothness terms enforce smoothness on two fields f and g, respectively. They involve a notion of neighborhood; we assume that there are two neighborhood systems: one on pixels

$$\mathcal{N}_1 \subset \{\{p, p'\} \mid p, p' \in \mathcal{P} \,, \;\; p \neq p'\}$$

and one on interactions

$$\mathcal{N}_2 \subset \{\{i, i'\} \mid i, i' \in I \,, \;\; i \neq i'\}.$$

$\mathcal{N}_1$ can be the usual 4-neighborhood system: pixels p and p' are neighbors if they are in the same image and $|p'_x - p_x| + |p'_y - p_y| = 1$. $\mathcal{N}_2$ can be defined similarly; interactions $\langle p, q, l\rangle$ and $\langle p', q', l\rangle$ are neighbors if p and p' are neighbors (or they are the same pixel): $|p'_x - p_x| + |p'_y - p_y| \leq 1$. The only requirement on $\mathcal{N}_2$ is that neighboring interactions must have the same depth label.

We will write the first smoothness term as

$$E^{(1)}_{smooth}(f) = \sum_{\{p,p'\}\in\mathcal{N}_1} V_{\{p,p'\}}(f(p), f(p'))$$

We will require the term $V_{\{p,q\}}$ to be a metric. This imposes smoothness while preserving discontinuities, as long as we pick an appropriate robust metric. For example, we can use the robustified L_1 distance $V(l_1, l_2) = \min(|l_1 - l_2|, R)$ for constant R. Note that this smoothness term involves only a single camera.

The second smoothness term can be written as

$$E^{(2)}_{smooth}(g) = \sum_{\{i,i'\}\in\mathcal{N}_2} V_{\{i,i'\}} \cdot T(g(i) \neq g(i'))$$

where $T(\cdot)$ is 1 if its argument is true and 0 otherwise. Note that this smoothness term involves pairs of cameras, as does [KZ01].

Visibility term

This term will encode the visibility constraint: it will be zero if this constraint is satisfied, and infinity otherwise. We can write this using another set of interactions I_{vis} which contains pairs of 3D-points violating the visibility constraint:

$$E_{visibility}(f) = \sum_{\langle p,f(p)\rangle,\langle q,f(q)\rangle\in I_{vis}} \infty$$

We require the set I_{vis} to meet following condition:

- Only 3D-points at different depths can interact, i.e. if $\{\langle p_1, l_1\rangle, \langle p_2, l_2\rangle\} \in I_{vis}$ then $l_1 \neq l_2$.

The visibility constraint says that if a 3D-point $\langle p, l\rangle$ is present in a configuration f (i.e. $l = f(p)$) then it "blocks" views from other cameras: if a ray corresponding to a pixel q goes through (or close to) $\langle p, l\rangle$ then its depth is at most l. Again, we need a definition of "closeness". We will use the set I for this purpose. Thus, the set I_{vis} can be defined as follows: it will contain all pairs of 3D-points $\langle p, l\rangle$, $\langle q, l'\rangle$ such that $\langle p, l\rangle$ and $\langle q, l\rangle$ interact (i.e. they are in I) and $l' > l$.

Consistency term

The last term will enforce consistency between two mappings f and g. It can be formulated as follows: if an interaction $\langle p, q, l\rangle$ is active, then the label for pixels p and q must be l. We can write this as

$$E_{consistency}(f, g) = \sum_{\langle p,q,l\rangle \in I} \infty \cdot T(g(\langle p, q, l\rangle) = 1 \ \wedge \ (f(p) \neq l \ \vee \ f(q) \neq l))$$

4.2.2 Discretization

An important issue is the choice of the sets of interactions I which basically defines a discretization. It can consist, for example, of pairs of 3D-points $\{\langle p, l\rangle, \langle q, l\rangle\}$ such that the distance in 3D-space between these points is less than some threshold. However, with this definition of closeness the number of interactions per pixel may vary greatly from pixel to pixel. Instead, we chose the following criterion: if p is a pixel in the image i, q is a pixel in the image j and $i < j$ then the 3D-points $\langle p, l\rangle$, $\langle q, l\rangle$ interact if the closest pixel to the projection of $\langle p, l\rangle$ onto the image j is q. Of course, we can include only interactions between certain images, for example, taken by neighboring cameras. Note that it makes sense to include interactions only between *different* cameras.

An example of our problem formulation in action is shown in figure 4.1. There are two cameras C1 and C2. There are 5 labels shown as black vertical lines. As

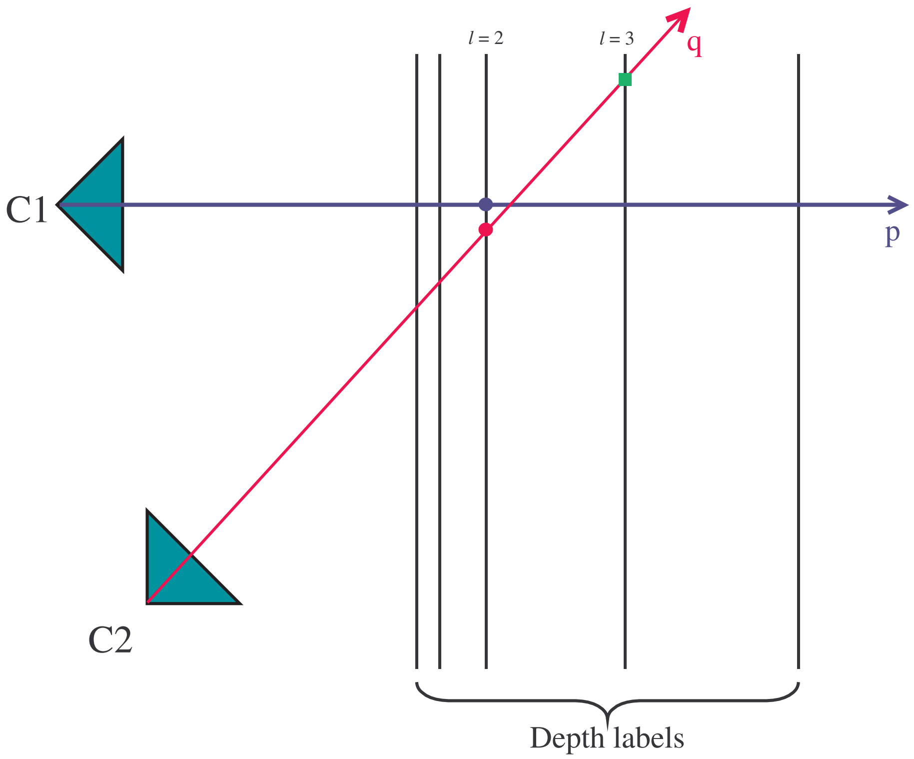


Figure 4.1: Example of pixel interactions. There is a photo-consistency constraint between the red round point and the blue round point. The red round point blocks camera C2's view of the green square point.

in our current implementation, labels are distributed by increasing distance from a fixed camera. Two pixels, p from C1 and q from C2 are shown, along with the red round 3D-point $\langle q, 2\rangle$ and the blue round 3D-point $\langle p, 2\rangle$. These points share the same label, and interact (i.e., $\{\langle p, 2\rangle, \langle q, 2\rangle\} = \langle p, q, 2\rangle \in I$). So there is a photoconsistency term between them. The green square point $\langle q, 3\rangle$ is at a different label (greater depth), but is behind the red round point. The pair of 3D-points $\{\langle p, 2\rangle, \langle q, 3\rangle\}$ is in I_{vis}. So if the ray p from camera C1 sees the red round point $\langle p, 2\rangle$, the ray q from C2 cannot see the green square point $\langle q, 3\rangle$.

4.3 Special cases

In this section we consider two special cases:

- I. Only the first smoothness term is present; $E^{(2)}_{smooth}$ is omitted.
- II. Only the second smoothness term is present; $E^{(1)}_{smooth}$ is omitted.

We will refer to cases I and II as "algorithm I" and "algorithm II", respectively. Algorithm I was originally presented in [KZ02a]. A special case of algorithm II (namely, the case of two rectified cameras) was presented in [KZ01].

4.3.1 Algorithm I

Let us consider our energy function with the second smoothness term $E^{(2)}_{smooth}$ omitted. We can view our energy as function of only one mapping f if we assume that g is determined from the minimality condition:

$$\tilde{E}(f) = E(f, g(f)) \quad \text{with} \quad g(f) = \arg\min_g E(f, g)$$

Let us consider an interaction $i = \langle p, q, l \rangle \in I$. Since g is not involved in the smoothness constraint, the value $g(i)$ will depend only on $f(p)$ and $f(q)$. Namely, if $f(p) \neq l$ or $f(q) \neq l$ then $g(i)$ must be 0 because of the consistency constraint between f and g. Now suppose that $f(p) = f(q) = l$. In this case the value $g(i)$ will be determined from the minimality condition: it will be 0 if $D_i(0) < D_i(1)$, and it will be 1 if $D_i(0) > D_i(1)$. Thus, the data term for interaction i becomes

$$\tilde{E}_{data(i)}(f) = D_i(0) + D(p, q) \cdot T(f(p) = f(q) = l)$$

where $D(p, q) = \min(D_i(1) - D_i(0), 0)$.

$D_i(0)$ is a constant which does not depend on configuration f; therefore, it can be omitted. Then the energy function becomes

$$E(f) = E^{(1)}_{data}(f) + E^{(1)}_{smooth}(f) + E_{vis}(f) \tag{4.2}$$

The terms $E^{(1)}_{smooth}$ and E_{vis} are defined above; the new term $E^{(1)}_{data}$ is

$$E^{(1)}_{data}(f) = \sum_{\langle p,q,l\rangle \in I} D(p,q) \cdot T(f(p) = f(q) = l)$$

where $D(p,q)$ is a non-positive number.

4.3.2 Algorithm II

Now let us consider our energy with the first smoothness term $E^{(1)}_{smooth}$ omitted. As before, we will view our energy as a function of only one mapping g (with f determined from the minimality condition). The energy function then becomes

$$E(g) = E_{data}(g) + E^{(2)}_{smooth}(g) + E^{(2)}_{vis}(g) \tag{4.3}$$

where E_{data} and $E^{(2)}_{smooth}$ are defined above and $E^{(2)}_{vis}(g)$ is a sum of two terms $E_{vis}(f(g)) + E_{consistency}(f(g), g)$. It is easy to see that this term enforces the following constraint: for every pair of 3D-points $\langle p, l\rangle$, $\langle p', l'\rangle$ in the visibility neighborhood I_{vis} interactions $\langle p, q, l\rangle$ and $\langle p', q', l'\rangle$ cannot both be active in g.

Algorithm II: case of two rectified cameras

Algorithm II can be formulated in a simpler form in one special case: the case of two rectified cameras C_{left} and C_{right}. Rectification means that corresponding pixels p and q lie on the same scanline, i.e. $p_y = q_y$.

Recall that the difference $q_x - p_x$ where p is a pixel in the left image and q is the corresponding pixel in the right image is called a *disparity* for the pixel p. Disparity directly depends on the distance from a camera to the object. If, for example, two cameras are aligned then disparity is inversely proportional to the distance to the object. Thus, we can use disparities as depth labels in our framework.

Then the set of interactions I can be represented by a collection of (unordered) pairs $\langle p, q \rangle$ where p, q are pixels in the left and the right cameras, respectively, that may potentially correspond. We omitted a depth label (i.e. disparity) l in this notation since it can be expressed from p and q as $l = q_x - p_x$.

The visibility term $E^{(2)}_{vis}$ effectively imposes the uniqueness: a pixel p can participate in at most one active interaction, i.e. it can correspond to at most one pixel in the other image.

Let us prove that the energy function can be written in the following form:

$$E(g) = \tilde{E}_{data}(g) + E_{occlusions}(g) + E^{(2)}_{smooth}(g) + E^{(2)}_{vis}(g) \tag{4.4}$$

where

- $\tilde{E}_{data}(g)$ is a modified data term:

  $$\tilde{E}_{data}(g) = \sum_{g(i)=1} D_i(g(i))$$

  (the summation is over all *active* interactions and does not include penalty K for inactive interactions)

- $E_{occlusions}(g)$ counts the number of occluded pixels: we add a penalty $2K$ for every occluded pixel in configuration g. (Recall that a pixel is occluded if it is not involved in any active interaction)

- $E^{(2)}_{smooth}(g)$ is a smoothness term on the mapping g (see above)

- $E^{(2)}_{vis}(g)$ enforces uniqueness constraint (see above)

Indeed, the only difference between equations 4.3 and 4.4 is that we replaced the term $E_{data}(g)$ with two terms $\tilde{E}_{data}(g) + E_{occlusions}(g)$. Let $N_0(g)$ and $N_1(g)$ be the number of inactive and active interactions in configuration g, respectively, and let $n_0(g)$ and $n_1(g)$ be the number of occluded and non-occluded pixels in g, respectively. Then $N_0(g)+N_1(g) = const$, $n_0(g)+n_1(g) = const$, $N_1(g) = 2n_1(g)$, so

$$E_{data}(g) = \tilde{E}_{data}(g) + K \cdot N_0(g) = const + \tilde{E}_{data}(g) + 2K \cdot n_0(g) =$$

$$= const + \tilde{E}_{data}(g) + E_{occlusions}(g)$$

as desired.

This formulation with energy 4.4 was first presented in [KZ01]. This paper used a slightly different terminology: interactions $\langle p, q\rangle \in I$ were called *assignments*.

4.3.3 Comparison of algorithms I and II

In this section we will present our intuition on differences between these two special cases. The major difference lies in the smoothness term. We argue that the smoothness term of algorithm I suits the problem more naturally. Indeed, the purpose of the smoothness term is to introduce a prior knowledge on the shape of object itself. Ideally, it should not depend on cameras observing this object. Unfortunately, a tractable formulation of the multicamera reconstruction problem with such a smoothness term is not known. Algorithm I uses the next best thing: the smoothness term depends on the shape of the object and on a position of one camera. ¿From this point of view, algorithm II is less natural since the smoothness term depends on the shape of the object and on positions of *two* cameras.

Another drawback of the second smoothness term is that it allows only piecewise constant depth maps, while the first smoothness term can be chosen to permit smooth depth variations (using, for example, robustified L_1 distance $V(l_1, l_2) = \min(|l_1 - l_2|, R)$ for constant R).

The difference between the two smoothness terms can be summarized as follows. There are two reasons why a smoothness constraint can be violated. The first reason is smooth depth variations which happen when the normal to the object's surface is not perpendicular to depth labels (planes). Algorithm I can handle this situation better than algorithm II since it allows using any smoothness term which is a metric. The second reason is depth changes at discontinuities (object boundaries). Again, algorithm I is more preferable. The smoothness constraint of algorithm I says that if a scene element is visible from a particular camera, then a close scene element at the same depth should also be visible from this camera. On the other hand, the smoothness constraint of algorithm II says that if a scene element is visible from two particular cameras, then a close scene element at the same depth should also be visible from these two cameras. The latter constraint seems to be violated more often than the former one.

Another difference between algorithms I and II is the representation used. Algorithm I represents the geometry of the scene by the mapping f which assigns a depth label to *every* pixel. A value of some assignments is questionable: some scene elements may be visible from only one of the cameras; in this case it is impossible to choose the correct depth label based on the photoconsistency constraint, so only smoothness constraint is used.

In contrast, algorithm II uses the mapping g to represent the geometry of the scene that marks which interactions are visible from both cameras. If there is not enough evidence that a pixel p has corresponding pixels in other images, then it won't participate in active interactions, so the depth label will be undetermined.

Pixel p will be marked as "occluded" which means that it is visible from only one of the cameras.

Occlusions are much more common when a small number of cameras is used (for example, two). Therefore, we argue that algorithm II has some merits over algorithm I in the case of two cameras (i.e. stereo case).

4.4 Graph construction

Exact minimization of energy function 4.1 and its special cases 4.2 and 4.3 is NP-hard, as shown in [KZ01, KZ02a]. Therefore, it is unlikely that there exists a polynomial time algorithm for computing the exact minimum.

We will present an approximate algorithm for minimizing the general form of the function E (equation 4.1) among all configurations. The algorithm relies on the notion of α-expansion introduced in [BVZ98, BVZ01] (see section 2.4). Since we use a different representation of the scene, we need to generalize this notion.

Consider an input configuration (f, g) and a disparity α. Another configuration (f', g') is defined to be within a single α-*expansion* of (f, g) if two conditions are satisfied:

- All pixels must either keep their depth labels, or change it to α. In other words, for any pixel $p \in \mathcal{P}$ either $f'(p) = f(p)$ or $f'(p) = \alpha$.

- All inactive interactions whose depth is different from α must remain inactive. In other words, for any interaction $\langle p, q, l \rangle \in I$ conditions $g(\langle p, q, l \rangle) = 0$ and $l \neq \alpha$ imply $g'(\langle p, q, l \rangle) = 0$.

Our algorithm is very straightforward; we simply select (in a fixed order or at random) a disparity α, and we find the configuration within a single α-expansion move (our local improvement step) that gives the largest decrease in the energy $E(f, g)$. If this decreases the energy, then we go there; if there is no α that decreases the energy, we are done. The output of our method will be a local minimum in a strong sense: no α-expansion move can decrease the energy. Except for the problem formulation and the choice of energy function, this algorithm is identical to the method of [BVZ01].

One restriction on the algorithm is that the initial configuration must satisfy the visibility and consistency constraints (i.e. the initial energy must be finite). This will guarantee that all subsequent configurations will have finite energies, i.e. they will satisfy these constraints as well.

The critical step in our method is to efficiently compute the α-expansion with the smallest energy. In this section, we show how to use graph cuts to solve this problem.

4.4.1 Energy minimization using graph cuts

Instead of doing an explicit problem reduction, we will use a result from chapter 3. We will show that the restriction of our energy function to configurations within a single expansion move results in a regular function of binary variables, i.e. a function of the form of equation 3.3. Note that it is not necessary to use only terms $E^{i,j}$ for which $i < j$ since we can swap the variables if necessary without affecting condition 3.4.

In an α-expansion, active interactions may become inactive, and inactive interactions whose depth is α may become active. Suppose that we start off with an initial configuration (f^0, g^0) satisfying the visibility and consistency constraints. The active interactions for a new configuration within one α-expansion will be a subset of $I^0 \cup I^\alpha$, where $I^0 = \{\langle p,q,l\rangle \in I \mid g^0(\langle p,q,l\rangle) = 1 \text{ and } l \neq \alpha\}$ and $I^\alpha = \{\langle p,q,\alpha\rangle \in I\}$.

It is easy to see that any configuration (f,g) within a single α-expansion of the initial configuration (f^0, g^0) can be encoded by two binary vectors $x = \{x_p \,|\, p \in \mathcal{P}\}$ and $y = \{y_i \mid i \in I^\alpha \cup I^0\}$. We will use the following formula for correspondence between binary vectors and configurations:

$$\begin{array}{rll}
\forall p \in \mathcal{P} & f(p) & = \begin{cases} f^0(p) & \text{if } x_p = 0 \\ \alpha & \text{if } x_p = 1 \end{cases} \\
\forall i \in I^0 & g(i) & = 1 - y_i \\
\forall i \in I^\alpha & g(i) & = y_i \\
\forall i \notin I^0 \cup I^\alpha & g(i) & = 0
\end{array}$$

Let us denote a configuration defined by vectors (x,y) as (f^x, g^y). We now have the energy of binary variables:

$$\tilde{E}(x,y) = \tilde{E}_{data}(y) + \tilde{E}^{(1)}_{smooth}(x) + \tilde{E}^{(2)}_{smooth}(y) + \tilde{E}_{vis}(x) + \tilde{E}_{consistency}(x,y)$$

where

$$\tilde{E}_{data}(y) = E_{data}(g^y),$$

$$\tilde{E}^{(1)}_{smooth}(x) = E^{(1)}_{smooth}(f^x),$$

$$\tilde{E}^{(2)}_{smooth}(y) = E^{(2)}_{smooth}(g^y),$$

$$\tilde{E}_{vis}(x) = E_{vis}(f^x),$$

$$\tilde{E}_{consistency}(x,y) = E_{consistency}(f^x, g^y).$$

We can now consider each term separately, and show that each is regular, i.e. satisfies condition (3.4).

1. Data term.

$$\tilde{E}_{data}(y) = \sum_{i \in I^0} D_i(1 - y_i) + \sum_{i \in I^\alpha} D_i(y_i)$$

Condition (3.4) is satisfied since each term in this sum depends only on one variable.

2. First smoothness term.

$$\tilde{E}^{(1)}_{smooth}(x) = \sum_{\{p,p'\} \in \mathcal{N}_1} V_{\{p,p'\}}(f^x(p), f^x(p')).$$

Let's consider a single term $E^{p,p'}(x_p, x_{p'}) = V_{\{p,p'\}}(f^x(p), f^x(p'))$. We assumed that $V_{\{p,p'\}}$ is a metric; thus, $V_{\{p,p'\}}(\alpha, \alpha) = 0$ and $V_{\{p,p'\}}(f(p), f(p')) \leq V_{\{p,p'\}}(f(p), \alpha) + V_{\{p,p'\}}(\alpha, f(p'))$, or $E^{p,p'}(1,1) = 0$ and $E^{p,p'}(0,0) \leq E^{p,p'}(0,1) + E^{p,p'}(1,0)$. Therefore, condition (3.4) holds.

3. Second smoothness term.

$$\tilde{E}^{(2)}_{smooth}(y) = \sum_{\{i,i'\} \in \mathcal{N}_2} V_{\{i,i'\}} \cdot T(g^y(i) \neq g^y(i'))$$

Let's consider a single term $E^{i,i'}(y_i, y_{i'}) = V_{\{i,i'\}} \cdot T(g^y(i) \neq g^y(i'))$. Since the depths of i and i' are the same, they either both belong to I^0 or both belong to I^α. In both cases condition $g^y(i) \neq g^y(i')$ is equivalent to condition $y_i \neq y_{i'}$. Thus, $E^{i,i'}(0,0) = E^{i,i'}(1,1) = 0$ and $E^{i,i'}(0,1) = E^{i,i'}(1,0) = V_{\{i,i'\}} \geq 0$, so condition (3.4) holds.

4. Visibility term.

$$\tilde{E}_{vis}(x) = \sum_{\langle p, f^x(p)\rangle, \langle q, f^x(q)\rangle \in I_{vis}} \infty$$

$$= \sum_{\langle p, l_p\rangle, \langle q, l_q\rangle \in I_{vis}} T(f^x(p) = l_p \wedge f^x(q) = l_q) \cdot \infty.$$

Let's consider a single term $E^{p,q}(x_p, x_q) = T(f^x(p) = l_p \wedge f^x(q) = l_q) \cdot \infty$. $E^{p,q}(0,0)$ must be zero since it corresponds to the visibility cost of the initial configuration and we assumed that the initial configuration satisfies the visibility constraint. Also

$E^{p,q}(1,1)$ is zero (if $x_p = x_q = 1$, then $f^x(p) = f^x(q) = \alpha$ and, thus, the conditions $f^x(p) = l_p$ and $f^x(q) = l_q$ cannot both be true since I_{vis} includes only pairs of 3D-points at different depths). Therefore, condition (3.4) holds since $E^{p,q}(0,1)$ and $E^{p,q}(1,0)$ are non-negative.

5. Consistency term.

$$\tilde{E}_{consistency}(x,y) = \sum_{\langle p,q,l\rangle \in I} \infty \cdot T(g^y(\langle p,q,l\rangle) = 1 \ \wedge \ (f^x(p) \neq l \ \vee \ f^x(q) \neq l))$$

The term involving interaction $i = \langle p,q,l\rangle$ can be rewritten as the sum $E^{p,i}(x_p, y_i)+$ $E^{q,i}(x_q, y_i)$ where $E^{p,i}(x_p, y_i) = \infty \cdot T(g^y(i) = 1 \ \wedge \ f^x(p) \neq l)$ and $E^{q,i}(x_q, y_i) = \infty \cdot T(g^y(i) = 1 \ \wedge \ f^x(q) \neq l)$. Let's consider one of the terms, for example $E^{p,i}$. Two cases are possible:

5A. $l \neq \alpha$. If $f^0(p) \neq l$ then $E^{p,i} \equiv 0$, otherwise $E^{p,i}(x_p, y_i) = \infty \cdot T(y_i = 0 \ \wedge \ x_p = 1)$, so $E^{p,i}(1,0) = \infty$ and $E^{p,i}(0,0) = E^{p,i}(1,1) = E^{p,i}(0,1) = 0$.

5B. $l = \alpha$. In this case $E^{p,i}(x_p, y_i) = \infty \cdot T(y_i = 1 \ \wedge \ x_p = 0)$, so $E^{p,i}(0,1) = \infty$ and $E^{p,i}(0,0) = E^{p,i}(1,1) = E^{p,i}(1,0) = 0$.

4.5 Experimental results

4.5.1 Implementational details

Depth labels

The planes for labels were chosen to be orthogonal to the main axis of the middle camera, and to increase with depth. The labels are distributed so that disparities depend on labels approximately linearly. The depth difference between planes was chosen in such a way that a label change by one results in disparity change by at most one both in x and y directions.

We selected the labels α in random order, and kept this order for all iterations. We performed three iterations. (The number of iterations until convergence was at most five but the result was practically the same). We started with an initial solution in which all pixel are assigned to the label having the largest depth.

Data term

For our data term D we made use of the method of Birchfield and Tomasi [BT98] to handle sampling artifacts, with a slight variation: we compute intensity intervals for each band (R,G,B) using four neighbors, and then take the average data penalty. (We used color images; results for grayscale images are slightly worse).

Smoothness terms

We used a simple Potts model for the first smoothness term:

$$V_{\{p,p'\}}(l_p, l_{p'}) = \begin{cases} U_{\{p,p'\}} & \text{if } l_p \neq l_{p'} \\ 0 & \text{otherwise} \end{cases}$$

The choice of $U_{\{p,p'\}}$ was designed to make it more likely that a pair of adjacent pixels in one image with similar intensities would end up with similar disparities. For pixels $\{p, p'\} \in \mathcal{N}_1$ the term $U_{\{p,p'\}}$ was implemented as an empirically selected decreasing function of $\Delta I(p, p')$ as follows:

$$U_{\{p,p'\}} = \begin{cases} 3\lambda_1 & \text{if } \Delta I(p, p') < 5 \\ \lambda_1 & \text{otherwise} \end{cases}$$

where $\Delta I(p, p')$ is the maximum value of $|Intensity(p) - Intensity(p')|$ for all three bands (R,G,B).

For the second smoothness term we used similar expression:

$$V_{\{i,i'\}} = \begin{cases} 3\lambda_2 & \text{if } \max(\Delta I(p, p'), \Delta I(q, q')) < 8 \\ \lambda_2 & \text{otherwise} \end{cases}$$

where $\langle p, q, l \rangle$, $\langle p', q', l \rangle$ are interactions i, i', respectively, and p and p' are pixels in the same image, as well as q and q'.

4.5.2 Choice of parameters

Our energy function as defined above depends on three numbers: penalty K for an interaction being passive and two regularization parameters λ_1 and λ_2. From our experience there is some range of these parameters in which results differ slightly, but quality is about the same. If, however, parameters are chosen very far from this range (e.g. several times off), then results start to degrade: they are either oversmoothed, or vice versa - too noisy. These ranges sometimes differ considerably for different datasets, which can probably be explained by different levels of noise in the images.

The problem of picking good parameters is inherent in many energy-based vision algorithms and is not very well understood. We have chosen some heuristics for picking these parameters automatically. Although they do no always give the best result, we believe that they usually avoid the two extremes mentioned above.

Our first heuristic concerns the choice of K. Let us consider algorithm I for simplicity. The data term then becomes $D(p, q) = \min(D_i(1) - K, 0)$ (it is added to the energy function only if both pixels p and q agree on the depth label). Consider too different extremes. If K is too small (e.g. smaller than all costs $D_i(1)$), then the penalty for a pixel p will be the same for all depth labels. If, on the other hand, K is too large, then all penalties (or, rather, encouragements) will be approximately $-K$, so the algorithm will try to maximize the *number* of active interactions without looking at their quality. Our approach is to try to balance these two effects: K should be the smallest number that discriminates, say, the best 25% matches for a pixel p. We arrive at the following algorithm:

- 1. For every pixel p compute $D(p)$ as the k-th smallest value of $D_i(1)$'s for all interactions i for a fixed pair of cameras involving pixel p, where k is 0.25 times the number of depth labels. We also require $k \geq 3$.

- 2. Set K to be the average of $D(p)$ for all pixels p.

The choice of regularization parameters λ_1 and λ_2 is less clear. Intuitively, smoothness and data terms should "balanced". Since the minimum data term is $-K$, we have decided to make these parameters proportional to K. We use the following constraint: $\lambda_1 + \lambda_2 = K/5$. Thus, parameters for algorithm I are $\lambda_1 = K/5, \lambda_2 = 0$, and parameters for algorithm II are $\lambda_1 = 0, \lambda_2 = K/5$.

4.5.3 Experiments: stereo

We have compared three algorithms: algorithm I, algorithm II and the method of [BVZ01] which was found to be the best algorithm for stereo according to [SZ99]. We have run these algorithms in two modes: without reporting occlusions and with reporting occlusions. Algorithm I produces depth maps for all pixels of both the left and the right images, therefore occlusions can be inferred by cross-checking: a pixel is occluded if it does not map back to itself. Algorithm II produces depth maps which already contain occlusions. We have filled occluded regions using some postprocessing: we have assigned to occluded pixels the depth label of the closest non-occluded left neighbor lying in the same scanline. Algorithm of [BVZ01] computes depths for all pixels in the left image. To produce occlusions, we have augmented the algorithm: we introduced a new label "occluded" with some fixed penalty.

Note that some applications may require depth maps for all pixels, while others would benefit if occlusions were explicitly marked since they provide important information about object boundaries.

We performed experiments on six stereo pairs. Their sizes and running times of algorithms I, II, I+II and [BVZ01] are given in the table below. These running times were obtained on 450MHz UltraSPARC II processor. The max flow implementation we used is one specifically designed for the kinds of graphs that arise in vision [BK01].

stereo pair	image	number	running times			
	size	of labels	I	II	I+II	[BVZ01]
Tsukuba	384 x 288	16	80 secs	69 secs	183 secs	35 secs
sawtooth	434 x 380	20	141 secs	115 secs	320 secs	66 secs
venus	434 x 383	22	159 secs	145 secs	394 secs	85 secs
map	284 x 216	30	68 secs	49 secs	176 secs	26 secs
meter	256 x 240	15	35 secs	26 secs	95 secs	14 secs
tree	256 x 233	8	18 secs	13 secs	43 secs	7 secs

First we evaluated these four algorithms in the mode without occlusions. We have computed the error statistics for the first four stereo pairs (these datasets have ground truth).

stereo pair	I	II	I+II	[BVZ01]
Tsukuba	5.91 (1.86)	5.82 (1.18)	6.64 (1.78)	7.17 (1.93)
sawtooth	11.77 (0.67)	12.13 (0.71)	11.55 (0.54)	11.86 (0.62)
venus	13.19 (0.69)	15.40 (1.07)	13.03 (0.58)	16.90 (0.75)
map	16.78 (6.53)	19.80 (6.78)	19.60 (6.69)	21.55 (6.84)

We determined the percentage of the pixels where the algorithm did not compute the correct disparity ("errors" - the first number), or a disparity within ± 1 of the correct disparity ("gross errors" - the second number). We counted only pixels that are not occluded according to the ground truth since depth labels of such pixels cannot be determined from the photoconsistency constraint.

As we mentioned above, algorithm II does not compute depth labels for all pixels - it marks some regions as "occluded". We have performed some postprocessing to fill these regions: we have assigned to occluded pixels the depth label of the closest non-occluded left neighbor lying in the same scanline.

We have also computed error statistics for Tsukuba stereo pair in the mode with reporting occlusions.

algorithm	Errors	Gross errors	False negatives	False positives
I	6.51%	2.66%	44.16%	1.03%
II	6.56%	2.17%	41.33%	1.33%
I+II	7.45%	2.91%	38.86%	1.41%
[BVZ01]	7.28%	2.14%	77.59%	0.62%

The first two columns count only pixels that are not occluded according to the ground truth. We considered labeling a pixel as occluded to be a gross error. The last two columns show error rates for occlusions.

4.5.4 Evaluation of two-camera stereo algorithms

So far we have compared our algorithms only with the algorithm of [BVZ01]. A much more extensive comparison of different stereo algorithms is given in [SS02][1]. Results of this comparison are shown in Figure 1.2. We now explain the statistics used.

[1]The comparison is available at http://www.middlebury.edu/stereo. As of September 1, 2003, it compares 25 algorithms.

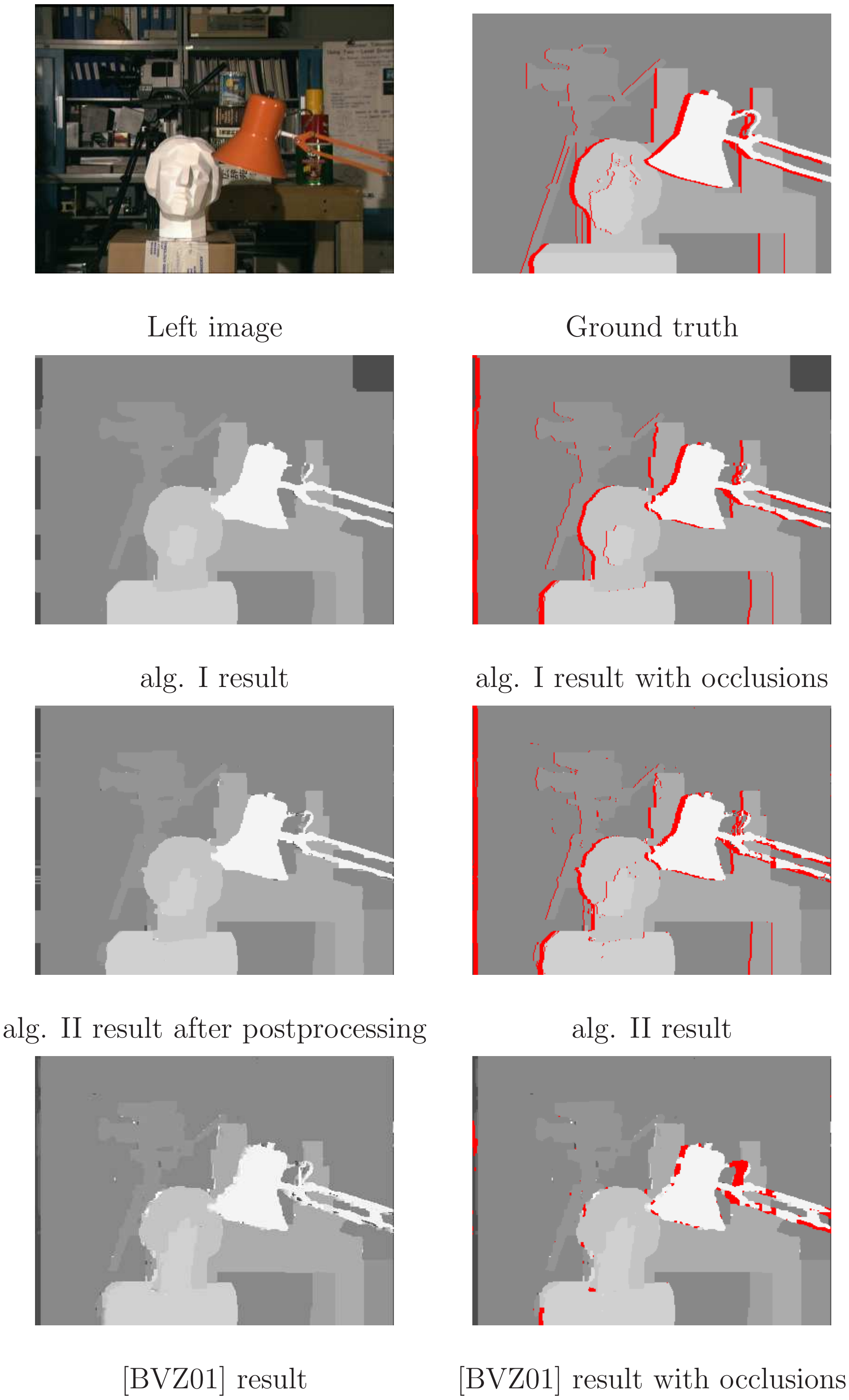
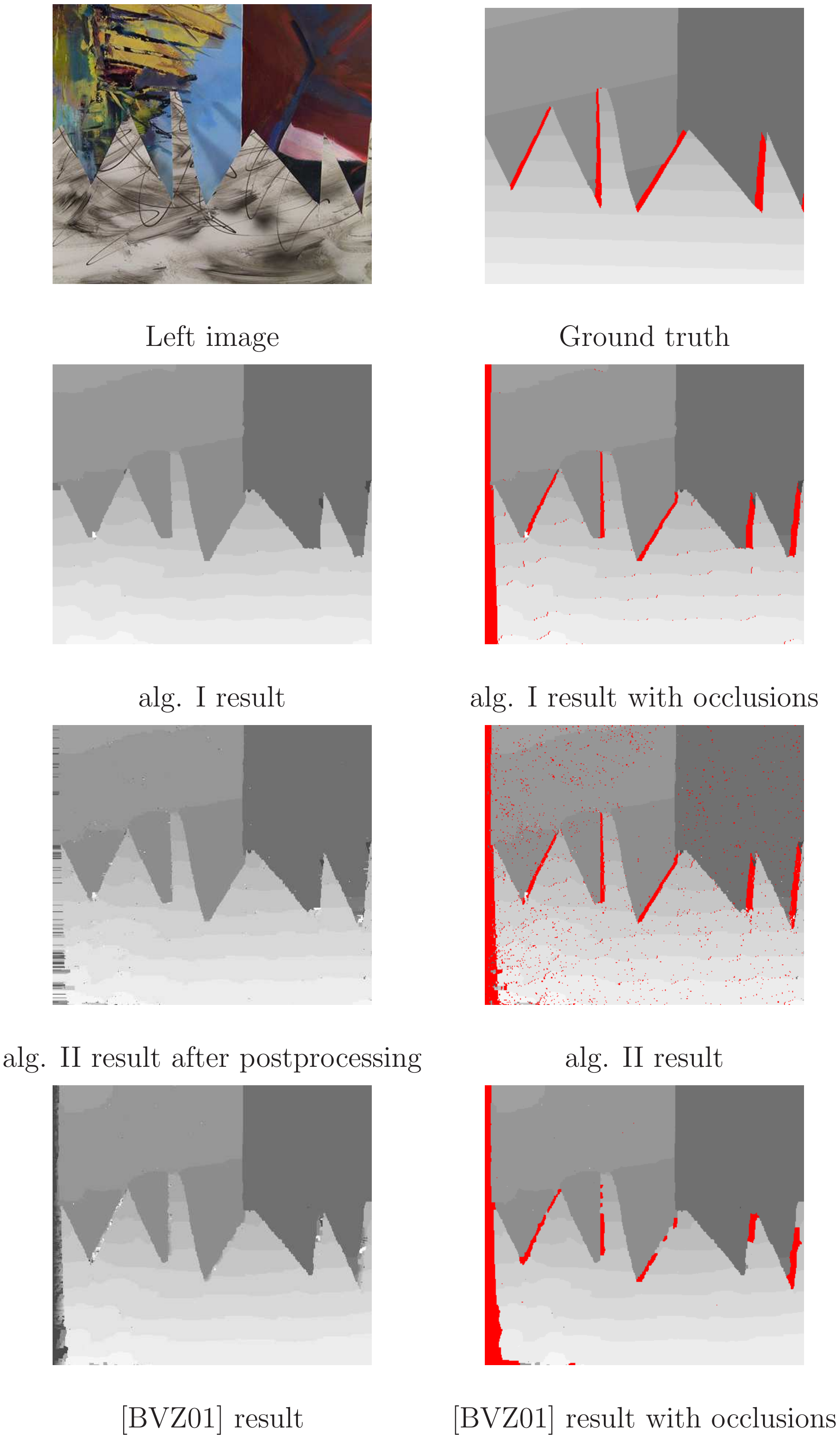


Figure 4.2: Stereo: results on Tsukuba pair.



Left image | Ground truth

alg. I result | alg. I result with occlusions

alg. II result after postprocessing | alg. II result

[BVZ01] result | [BVZ01] result with occlusions

Figure 4.3: Stereo: results on sawtooth pair.

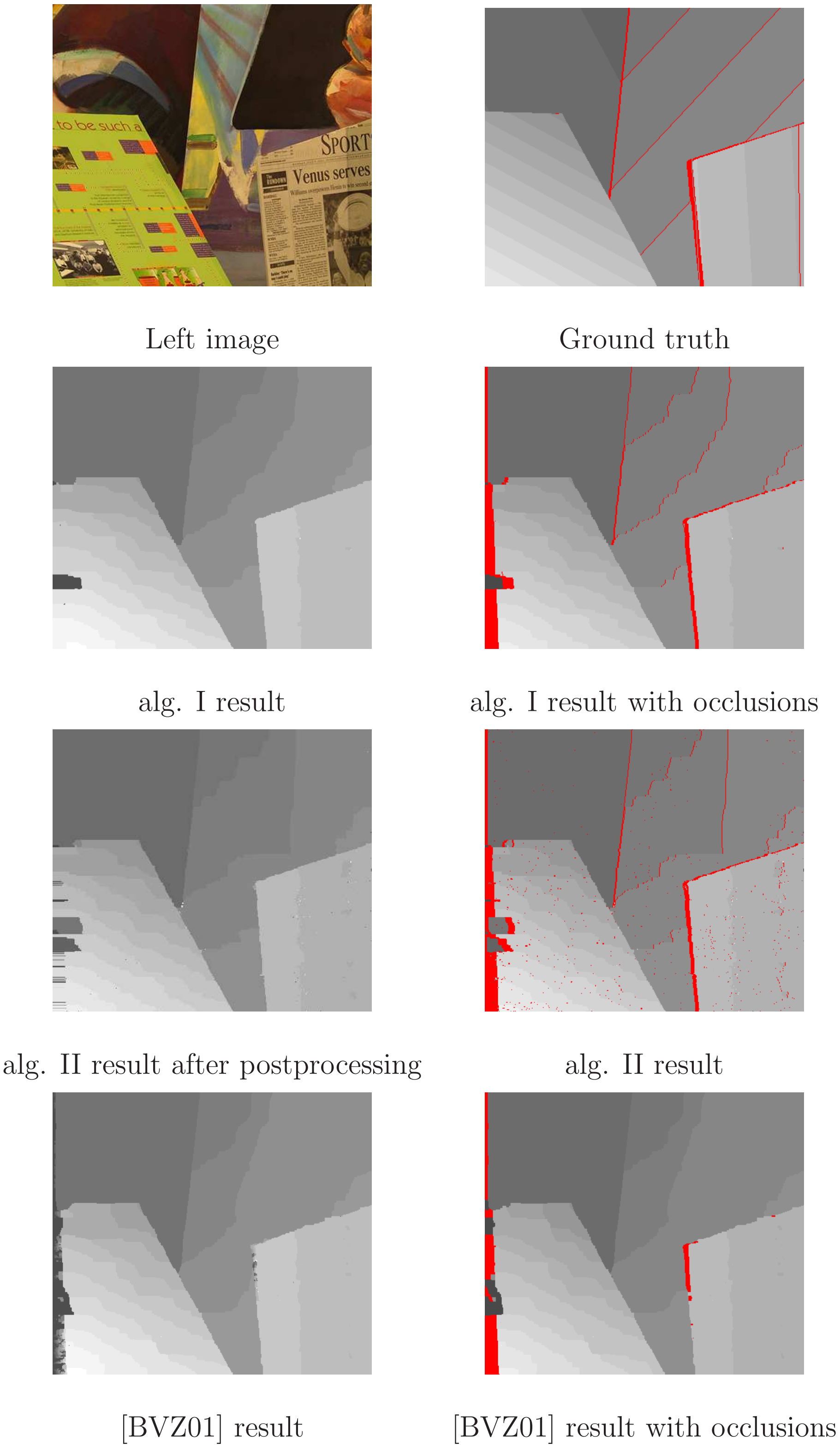
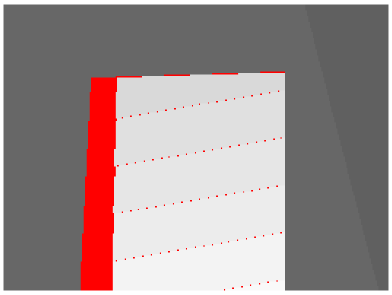


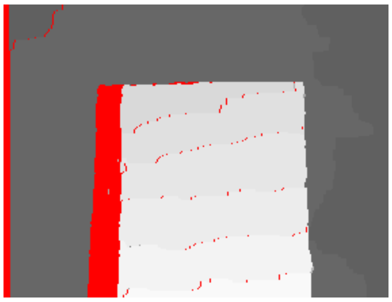
Figure 4.4: Stereo: results on venus pair.

Left image



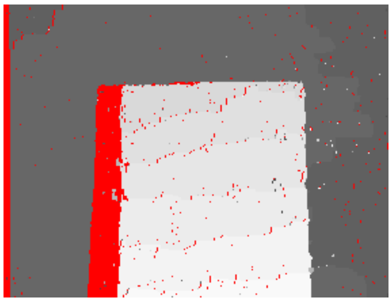
Ground truth

alg. I result

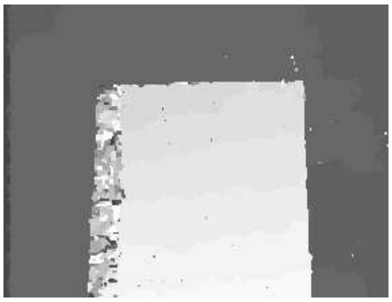


alg. I result with occlusions

alg. II result after postprocessing



alg. II result



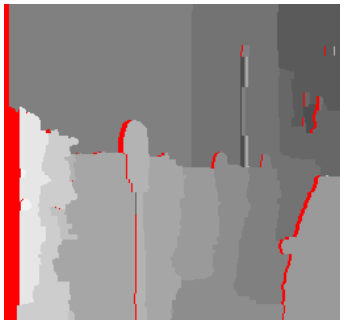
[BVZ01] result

[BVZ01] result with occlusions

Figure 4.5: Stereo: results on map pair.

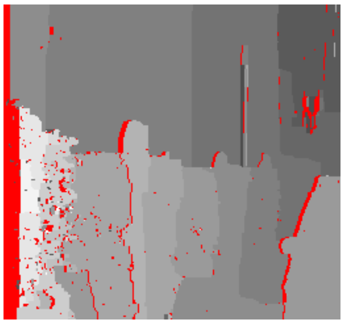
Left image

alg. I result



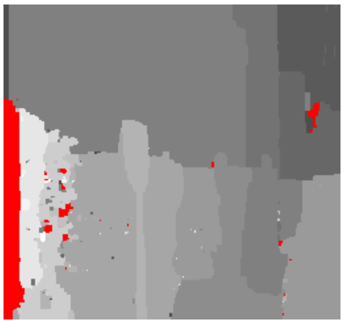
alg. I result with occlusions

alg. II result after postprocessing



alg. II result

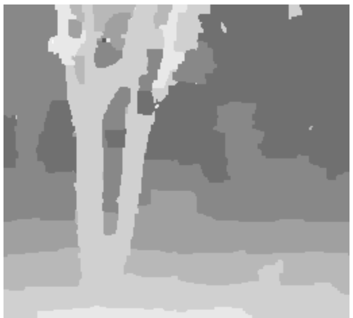
[BVZ01] result



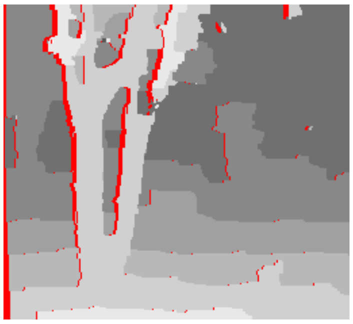
[BVZ01] result with occlusions

Figure 4.6: Stereo: results on meter pair.

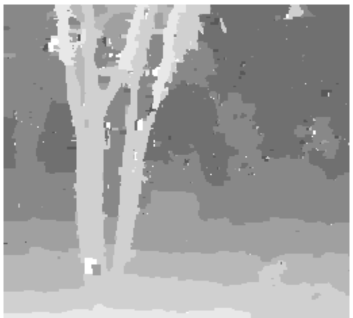
Left image



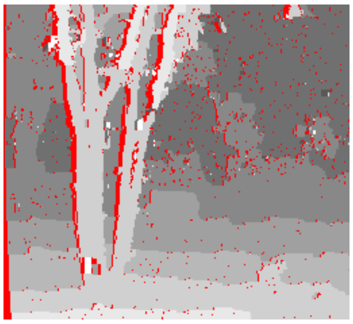
alg. I result



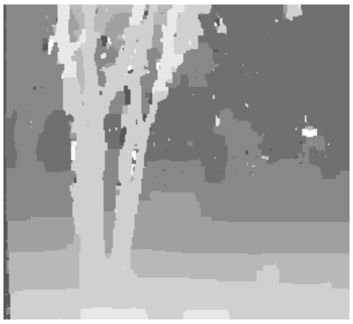
alg. I result with occlusions



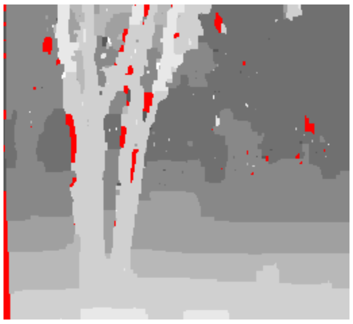
alg. II result after postprocessing



alg. II result



[BVZ01] result



[BVZ01] result with occlusions

Figure 4.7: Stereo: results on tree pair.

Stereo algorithms are evaluated on four datasets with ground truth: Tsukuba, Sawtooth, Venus and Map. For each dataset three numbers are computed: error percentage for the entire image (excluding occluded pixels), error percentage for pixels in untextured regions and error percentage for pixels near disparity discontinuities. One exception is the Map dataset where almost all pixels are in textured regions. A disparity is considered to be incorrect if it differs by more than 1 from the true disparity.

The small number indicate the rank of each algorithm in each column. The overall rank for an algorithm is determined as the sum of ranks for all columns.

Note that algorithm II is included twice in this table. The algorithm ranked #7 is our old submission without the heuristics for choosing parameters. The algorithm of [BVZ01] listed under the name "Graph cuts" is also included twice. These two submissions are due to different authors with slighly different implementational details.

Algorithms developed in this thesis have a sufficiently high ranking - #2 and #4. However, this does not necessarily mean that they are worse than algorithm #1 and better than algorithm #5. Different authors used different implementational details, such as different parameters, different data and smoothness costs; some authors use grayscale images, others use color images. From our experience, these details can have a big impact on performance.

4.5.5 Experiments: multiple cameras

We performed experiments on three datasets: the 'head and lamp' image from Tsukuba University, the flower garden sequence and the Dayton sequence. We used 5 images for the Tsukuba dataset (center, left, right, top, bottom), 8 images for the flower garden sequence and 5 images for the Dayton sequence. For the Tsukuba dataset we performed two experiments: in the first one we used 4 pairs of interacting cameras (center-left, center-right, center-top, center-bottom), and in the second one (marked with an asterisk) we used all 10 possible pairs. For the flower garden and Dayton sequences we used 7 and 4 interacting pairs, respectively (only adjacent cameras interact). The table below show image dimensions and running times obtained on 450MHz UltraSPARC II processor.

dataset	number of images	image size	running times I	running times II	running times I+II
Tsukuba	5	384 x 288	366 secs	707 secs	760 secs
Tsukuba*	5	384 x 288	774 secs	3794 secs	1771 secs
Flower garden	8	352 x 240	850 secs	929 secs	1078 secs
Dayton	5	384 x 256	543 secs	520 secs	664 secs

For algorithms I and II we used the parameters described in section 4.5.2. For the case I+II we used the following parameters: $\lambda_1 = \lambda_2 = K/10$. Note that cases I and II are usually faster than the case I+II since in the former case graphs that we construct contain only nodes corresponding to one of the mappings f or g. Also note that for the "Tsukuba*" dataset case II takes longer than case I+II. This is due to an inefficient implementation[2] of algorithm II.

We have computed the error statistics for the Tsukuba dataset, which are shown in the table below.

dataset	I	II	I+II
Tsukuba	5.93 (2.76)	6.39 (2.83)	5.95 (2.72)
Tsukuba*	4.66 (2.25)	5.19 (2.19)	5.07 (2.24)

[2]when interacting pairs of cameras form cycles, hard visibility constraints are enforced twice

As before, the first number denotes the percentage of errors and the second - the percentage of gross errors. However, unlike the stereo case, we counted all pixels in the image.

The images are shown in figure 4.8. The image at bottom right shows the areas where algorithm I differs from ground truth (black is no difference, gray is a difference of ± 1, and white is a larger difference). Inspecting the image shows that we in general achieve greater accuracy at discontinuities than results obtained from two views (see below); for example, the camera in the background and the lamp are more accurate. The major weakness of our output is in the top right corner, which is an area of low texture. The behavior of our method in the presence of low texture needs further investigation.

Figures 4.9 and 4.10 show our reconstructions for the two other datasets. Note that results for algorithm II contain scattered pixels marked in red. These are pixels with no depth labels (or, more precisely, assigning any depth label to such pixels results in the same value of the energy function). According to the formulation of algorithm II, this should mean that such pixels are visible only in one camera. However, inspecting the images shows that it is not the case: unlike stereo, all scene elements are visible from at least two cameras in our datasets. These occluded pixels are probably due to a high level of noise in these datasets (or their miscalibration). We performed some postprocessing of the results to fill these regions: we assign to such pixels the label of the closest labeled pixel. Note that this postprocessing is different from the technique in the previous section since that technique was specific to the stereo case.

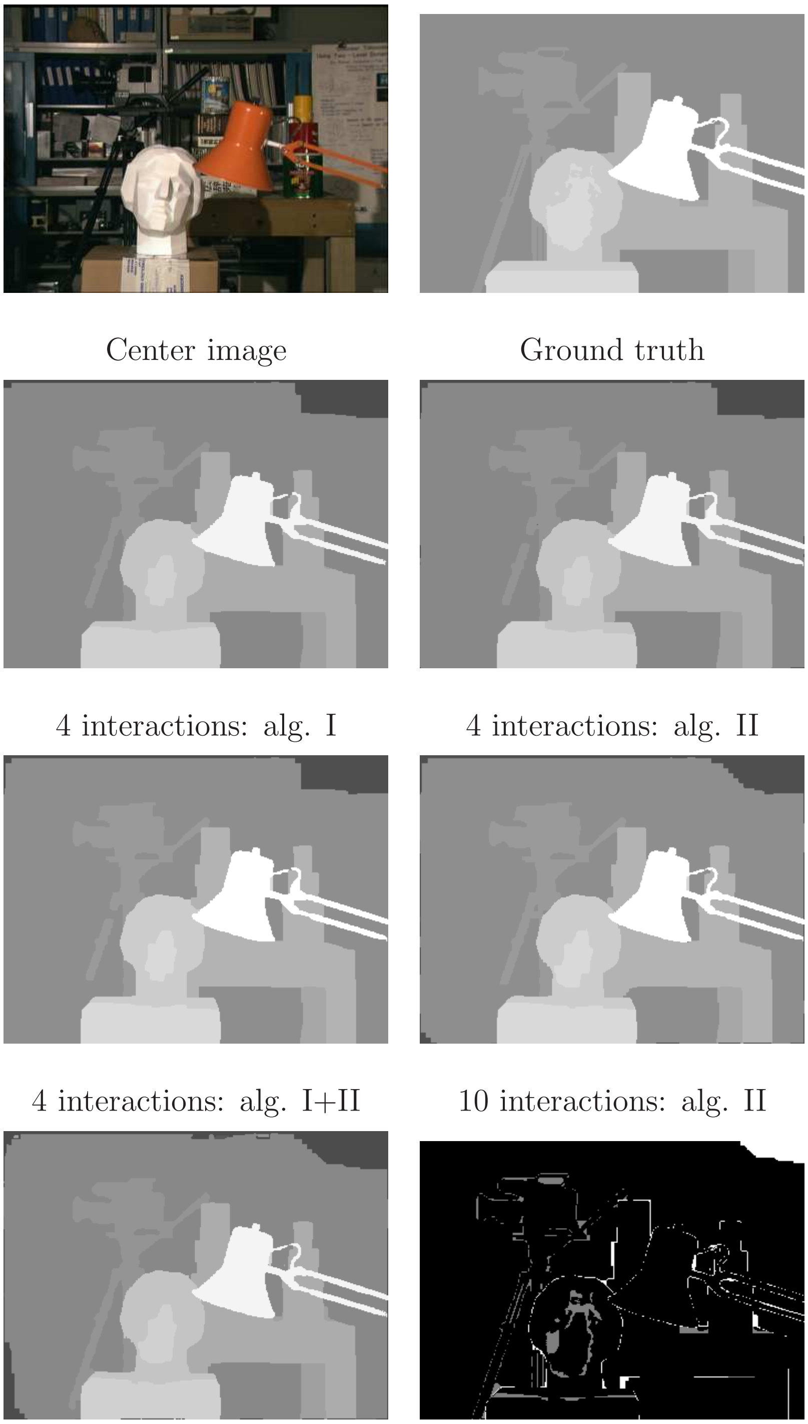
Parameter sensitivity

Experiments described so far were performed using automatically selected parameters (see section 4.5.2). We have also experimented with the parameter sensitivity of our method for the Tsukuba dataset using 4 pairs of interacting cameras. The tables below show that our method is relatively insensitive to the exact choice of K, λ_1 and λ_2. (Note that our selection procedure gives $K = 15$, $\lambda_1 + \lambda_2 = 3$).

K=10	λ_1=0	λ_1=1	λ_1=3	λ_1=9
λ_2=0	7.20 (2.77)	5.29 (2.67)	6.55 (2.93)	9.34 (2.91)
λ_2=1	5.66 (2.73)	5.86 (2.72)	8.34 (2.98)	9.38 (2.92)
λ_2=3	6.86 (2.98)	8.16 (2.93)	7.44 (1.94)	9.13 (2.87)
λ_2=9	8.27 (1.94)	8.75 (2.41)	8.76 (2.39)	10.37 (2.95)

K=30	λ_1=0	λ_1=1	λ_1=3	λ_1=9
λ_2=0	6.86 (2.72)	6.14 (3.23)	6.30 (2.88)	8.67 (3.59)
λ_2=1	6.24 (3.05)	6.10 (2.78)	6.13 (2.74)	8.85 (3.61)
λ_2=3	6.35 (2.88)	6.27 (2.83)	8.71 (3.46)	9.04 (3.58)
λ_2=9	7.35 (1.93)	7.34 (1.91)	7.46 (1.92)	7.68 (1.91)

K=50	λ_1=0	λ_1=1	λ_1=3	λ_1=9
λ_2=0	7.24 (2.81)	6.48 (3.34)	6.20 (2.89)	8.60 (3.58)
λ_2=1	6.66 (3.48)	6.47 (3.07)	6.29 (2.85)	8.71 (3.64)
λ_2=3	6.40 (2.92)	6.49 (2.86)	8.11 (3.47)	8.76 (3.62)
λ_2=9	8.85 (3.61)	8.97 (3.63)	9.01 (3.63)	9.20 (3.63)



Center image

Ground truth

4 interactions: alg. I

4 interactions: alg. II

4 interactions: alg. I+II

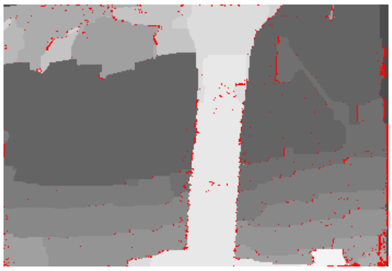
10 interactions: alg. II

10 interactions: alg. I

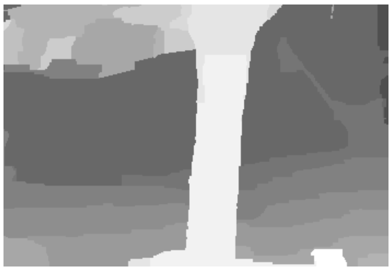
Comparison of alg. I result with ground truth

Figure 4.8: Results on Tsukuba dataset (5 images).

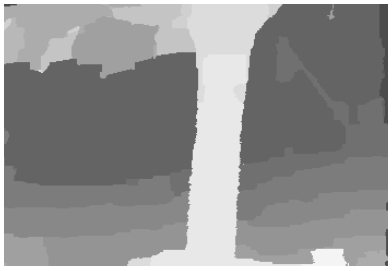
Middle image



alg. II result

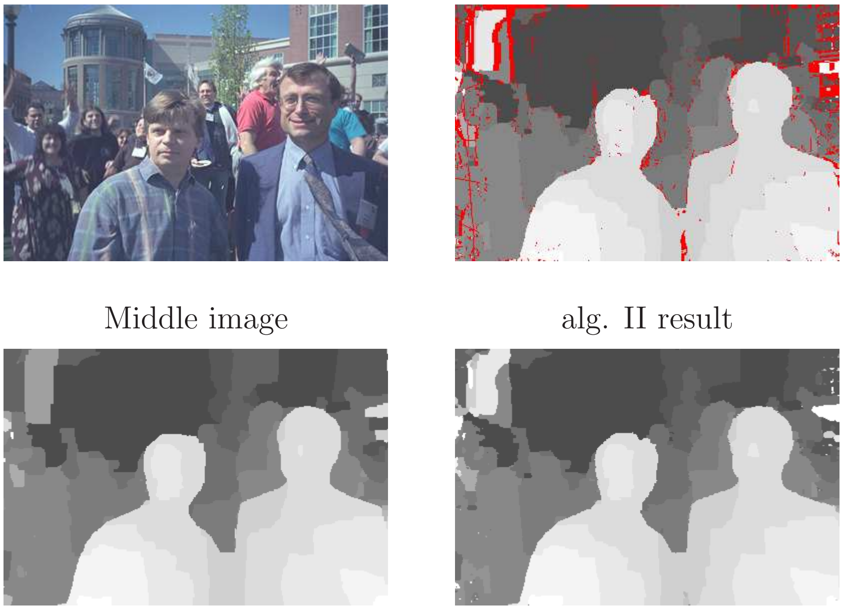


alg. I result



alg. II result after postprocessing

Figure 4.9: Results on the flower garden sequence (8 images).



Middle image | alg. II result

alg. I result | alg. II result after postprocessing

Figure 4.10: Results on the Dayton sequence (5 images).

4.6 Conclusions

To summarize, all algorithms I, II and I+II show very promising results and are quite comparable. If many views are used, then the difference between them is negligible. In the case of stereo conclusions are not so clear. Algorithm I generally seems to perform better, especially for generating depth maps without occlusions. However there are cases when algorithm II gives better results.

Algorithm I+II is slower than algorithms I or II, uses more memory since it constructs bigger graphs, and does not give any clear advantages over algorithms I or II. Therefore, it is unlikely to be used in practice.

Chapter 5

Maxflow algorithms for computer vision

5.1 Introduction

A growing number of publications in vision use graph cut energy minimization techniques for applications like image segmentation, restoration, stereo reconstruction, voxel occupancy, object recognition, augmented reality, texture synthesis, and others (see section 1.2). The graphs corresponding to these applications are usually huge 2D or 3D grids, and min-cut/max-flow algorithm efficiency is an issue that cannot be ignored.

The goal of this chapter is to examine the performance of several min-cut/max-flow algorithms on graphs typical for applications in vision, including a new method that we developed especially for these applications. The chapter is organized as follows. Section 5.2 introduces a new min-cut/max-flow algorithm that we developed while working with graphs in vision. In section 5.3 we tested our new algorithm and three standard min-cut/max-flow algorithms: the H_PRF and Q_PRF versions of the Goldberg-style "push-relabel" method [GT88, CG97], and Dinic's algorithm [Din70]. We selected several examples in image restoration, stereo, and segmentation where different forms of energy (1.1) are minimized via graph structures originally described in [GPS89, IG98b, BVZ98, BVZ01, KZ01, KZ02a, BJ01]. Such (or very similar) graphs are used in all computer vision papers known to us that use graph cut algorithms. In many interesting cases our new algorithm was significantly faster than the standard min-cut/max-flow techniques from combinatorial optimization. More detailed conclusions are presented in section 5.4.

5.1.1 Standard combinatorial optimization algorithms

An important fact in combinatorial optimization is that there are polynomial algorithms for min-cut/max-flow problems on directed weighted graphs with two terminals. These algorithms can be divided into two main groups: Goldberg-Tarjan style "push-relabel" methods [GT88] and algorithms based on Ford-Fulkerson style augmenting paths [FF62]. There are also some randomized algorithms (e.g. [Kar99]) for min-cut/max-flow problems but they are suitable only for dense undirected graphs which are not relevant for energy minimization in vision.

Standard augmenting paths based algorithms, such as Dinic's algorithm [Din70], work by pushing flow along non-saturated paths from the source to the sink until the maximum flow in the graph $\mathcal{G}$ is reached. A typical augmenting path algorithm stores information about the distribution of the current $s \to t$ flow f among the edges of $\mathcal{G}$ using a *residual graph* $\mathcal{G}_f$. The topology of $\mathcal{G}_f$ is identical to $\mathcal{G}$ but capacity of an edge in $\mathcal{G}_f$ reflects the residual capacity of the same edge in $\mathcal{G}$ given the amount of flow already in the edge. At the initialization there is no flow from the source to the sink (f=0) and edge capacities in the residual graph $\mathcal{G}_0$ are equal to the original capacities in $\mathcal{G}$. At each new iteration the algorithm finds the shortest $s \to t$ path along non-saturated edges of the residual graph. If a path is found then the algorithm *augments* it by pushing the maximum possible flow df that saturates at least one of the edges in the path. The residual capacities of edges in the path are reduced by df while the residual capacities of the reverse edges are increased by df. Each augmentation increases the total flow from the source to the sink $f = f + df$. The maximum flow is reached when any $s \to t$ path crosses at least one saturated edge in the residual graph $\mathcal{G}_f$.

Dinic's algorithm uses breadth-first search to find the shortest paths from s to t on the residual graph $\mathcal{G}_f$. After all shortest paths of a fixed length k are saturated, the algorithm starts the breadth-first search for $s \to t$ paths of length $k+1$ from scratch. Note that the use of shortest paths is an important factor that improves running time complexities for algorithms based on augmenting paths. The worst case running time complexity for Dinic's algorithm is $O(mn^2)$ where n is the number of nodes and m is the number of edges in the graph.

Push-relabel algorithms [GT88] use quite a different approach. They do not maintain a valid flow during the operation; there are "active" nodes that have a positive "flow excess". Instead, the algorithms maintain a labeling of nodes giving a low bound estimate on the distance to the sink along non-saturated edges. The algorithms attempt to "push" excess flows towards nodes with smaller estimated distance to the sink. Typically, the "push" operation is applied to active nodes with the largest distance (label) or based on FIFO selection strategy. The distances (labels) progressively increase as edges are saturated by push operations. Undeliverable flows are eventually drained back to the source. We recommend our favorite text-book on basic graph theory and algorithms [CCPS98] for more details on push-relabel and augmenting path methods.

5.2 A new min-cut/max-flow algorithm for vision

In this section we present a new algorithm that we developed while working with graphs that are typical for energy minimization methods in computer vision. The algorithm presented here belongs to the group of algorithms based on augmenting paths. Similarly to Dinic [Din70] it builds search trees for detecting augmenting paths. In fact, we build two search trees, one from the source and the other from the sink[1]. The other difference is that we reuse these trees and never start building them from scratch. The drawback of our approach is that the augmenting paths found are not necessarily the shortest augmenting paths; thus the time complexity of the shortest augmenting path is no longer valid. The trivial upper bound on the number of augmentations for our algorithm is the cost of the minimum cut $|C|$, which results in the worst case complexity $O(mn^2|C|)$. Theoretically speaking, this is worse than complexity of the standard algorithms discussed in section 5.1.1. However, experimental comparison in section 5.3 shows that on typical problem instances in vision our algorithm significantly outperforms standard algorithms.

5.2.1 Algorithm overview

Figure 5.1 illustrates our basic terminology. We maintain two non-overlapping search trees S and T with roots at the source s and the sink t, correspondingly. In tree S all edges from each parent node to its children are non-saturated, while in tree T edges from children to their parents are non-saturated. The nodes that are not in S or T are called "free". We have

$$S \subset \mathcal{V}, \quad s \in S, \quad T \subset \mathcal{V}, \quad t \in T, \quad S \cap T = \emptyset.$$

The nodes in the search trees S and T can be either "active" or "passive". The active nodes represent the outer border in each tree while the passive nodes are internal. The point is that active nodes allow trees to "grow" by acquiring new

[1]Note that in an earlier publication [BK01] we used a single tree rooted at the source that searched for the sink. The two-trees version presented here treats the terminals symmetrically. Experimentally, the new algorithm consistently outperforms the one in [BK01].

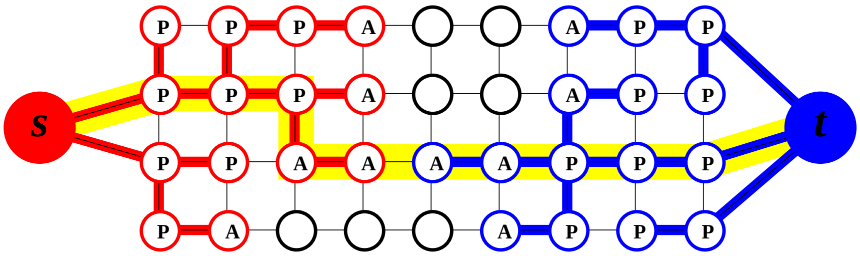


Figure 5.1: Example of the search trees S (red nodes) and T (blue nodes) at the end of the growth stage when a path (yellow line) from the source s to the sink t is found. Active and passive nodes are labeled by letters `A` and `P`, correspondingly. Free nodes appear in black.

children (along non-saturated edges) from a set of free nodes. The passive nodes can not grow as they are completely blocked by other nodes from the same tree. It is also important that active nodes may come in contact with the nodes from the other tree. An augmenting path is found as soon as an active node in one of the trees detects a neighboring node that belongs to the other tree.

The algorithm iteratively repeats the following three stages:

- "growth" stage: search trees S and T grow until they touch giving an $s \to t$ path
- "augmentation" stage: the found path is augmented, search tree(s) brake into forest(s)
- "adoption" stage: trees S and T are restored.

At the growth stage the search trees expand. The active nodes explore adjacent non-saturated edges and acquire new children from a set of free nodes. The newly acquired nodes become active members of the corresponding search trees. As soon as all neighbors of a given active node are explored the active node becomes passive. The growth stage terminates if an active node encounters a neighboring node that belongs to the opposite tree. In this case we detect a path from the source to the sink, as shown in Figure 5.1.

The augmentation stage augments the path found at the growth stage. Since we push through the largest flow possible some edge(s) in the path become saturated. Thus, some of the nodes in the trees S and T may become "orphans", that is, the edges linking them to their parents are no longer valid (they are saturated). In fact, the augmentation phase may split the search trees S and T into forests. The source s and the sink t are still roots of two of the trees while orphans form roots of all other trees.

The goal of the adoption stage is to restore single-tree structure of sets S and T with roots in the source and the sink. At this stage we try to find a new valid parent for each orphan. A new parent should belong to the same set, S or T, as the orphan. A parent should also be connected through a non-saturated edge. If there is no qualifying parent we remove the orphan from S or T and make it a free node. We also declare all its former children as orphans. The stage terminates when no orphans are left and, thus, the search tree structures of S and T are restored. Since some orphan nodes in S and T may become free the adoption stage results in contraction of these sets.

After the adoption stage is completed the algorithm returns to the growth stage. The algorithm terminates when the search trees S and T can not grow (no active nodes) and all edges leaving S are saturated, as well as edges entering T. This implies that a maximum flow is achieved. Corresponding minimum cuts can be determined either as $\mathcal{S} = S$, $\mathcal{T} = \mathcal{V} - S$, or as $\mathcal{S} = \mathcal{V} - T$, $\mathcal{T} = T$.

5.2.2 Details of implementation

Assume that we have a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. As for any augmenting path algorithm, we will maintain a flow f and the residual graph G_f (see section 5.1.1). We will keep the lists of all active nodes, A, and all orphans, O. The general structure of the algorithm is:

```
initialize:  S = {s}, T = {t}, A = {s,t}, O = ∅
while true
            grow S or T to find an augmenting path P from s to t
            if P = ∅ terminate
            augment on P
            adopt orphans
end while
```

The details of the *growth*, *augmentation*, and *adoption* stages are described below. It is convenient to store content of search trees S and T via flags $TREE(p)$ indicating affiliation of each node p so that

$$TREE(p) = \begin{cases} S & if\ p \in S \\ T & if\ p \in T \\ \emptyset & if\ p\, is free \end{cases}$$

If node p belongs to one of the search trees then the information about its parent will be stored as $PARENT(p)$. Roots of the search trees (the source and the sink), orphans, and all free nodes have no parents, t.e. $PARENT(p) = \emptyset$. We will also use notation $tree\_cap(p \to q)$ to describe residual capacity of either edge (p, q) if $TREE(p) = S$ or edge (q, p) if $TREE(p) = T$. These edges should be non-saturated in order for node p to be a valid parent of its child q depending on the search tree.

Growth stage:

At this stage active nodes acquire new children from a set of free nodes.

```
while A ≠ ∅
   pick an active node p ∈ A
   for every neighbor q such that tree_cap(p → q) > 0
      if TREE(q) = ∅ then add q to search tree as an active node:
          TREE(q) := TREE(p), PARENT(q) := p, A := A ∪ {q}
      if TREE(q) ≠ ∅ and TREE(q) ≠ TREE(p) return P = PATH_{s→t}
   end for
   remove p from A
end while
return P = ∅
```

Augmentation stage:

The input for this stage is a path P from s to t. Note that the orphan set is empty in the beginning of the stage, but there might be some orphans in the end since at least one edge in P becomes saturated.

```
find the bottleneck capacity Δ on P
update the residual graph by pushing flow Δ through P
for each edge (p, q) in P that becomes saturated
    if TREE(p) = TREE(q) = S
    then set PARENT(q) := ∅ and O := O ∪ {q}
    if TREE(p) = TREE(q) = T
    then set PARENT(p) := ∅ and O := O ∪ {p}
end for
```

Adoption stage:

During this stage all orphan nodes in O are processed until O becomes empty. Each node p being processed tries to find a new valid parent within the same search tree; in case of success p remains in the tree but with a new parent, otherwise it becomes a free node and all its children are added to O.

```
while O ≠ ∅
    pick an orphan node p ∈ O and remove it from O
    process p
end while
```

The operation "`process` p" consists of the following steps. First we are trying to find a new valid parent for p among its neighbors. A valid parent q should satisfy: $TREE(q) = TREE(p)$, $tree\_cap(q \to p) > 0$, and the "origin" of q

should be either source or sink. Note that the last condition is necessary because during adoption stage some of the nodes in the search forests S or T may originate from orphans.

If node p finds a new valid parent q then we set $PARENT(p) = q$. In this case p remains in its search tree and the active (or passive) status of p remains unchanged. If p does not find a valid parent then p becomes a free node and the following operations are performed:

- scan all neighbors q of p such that $TREE(q) = TREE(p)$:

  – if $tree\_cap(q \rightarrow p) > 0$ add q to the active set A

  – if $PARENT(q) = p$ add q to the set of orphans O and set $PARENT(q) := \emptyset$

- $TREE(p) := \emptyset, \;\; A := A - \{p\}$

Note that as p becomes free all its neighbors connected through non-saturated edges should became active. It may happen that some neighbor q did not qualify as a valid parent during adoption stage because it did not originate from the source or the sink. However, this node could be a valid parent after adoption stage is finished. At this point q must have active status as it is located next to a free node p.

5.2.3 Correctness proof

Let's introduce some invariants which are maintained during the execution of the algorithm.

I1 • *S and T are disjoint sets*

- *S is a forest with roots at either the source or orphans.*

- *T is a forest with roots at either the sink or orphans.*

I2 *If nodes p and q belong to the same search forest (S or T) then the residual capacity $tree\_cap(p \to q)$ is positive.*

I3 *There are no orphans during the growth stage.*

I4 *For passive nodes p in S or T the following property holds: if a neighbor q of p does not belong to the same search forest as p (i.e. $TREE(q) \neq TREE(p)$) then the residual capacity $tree\_cap(p \to q)$ must be zero.*

These invariants are clearly true at the initialization of the algorithm. It is easy to see these invariants directly follow from the construction of the algorithm.

Let's show that all stages terminate. The growth stage terminates because the number of nodes is finite. The same argument applies to the augmentation stage. Now we prove that the adoption stage is also finite. Note that after a node p in O has been processed it can not become an orphan again during the same adoption stage (it will imply that the adoption stage terminates after processing at most n nodes). Indeed, if p is moved from S or T to the set of free nodes then this holds since free nodes are not involved at the adoption stage. Suppose p found a new parent q and remained in the search forest. The new parent q must originate from a terminal (the source or the sink, depending on the search forest). Thus, the same terminal is the new origin of p as well. By construction, only descendants of orphans may become orphans during the adoption stage. Therefore, p can not become an orphan again at the same adoption stage.

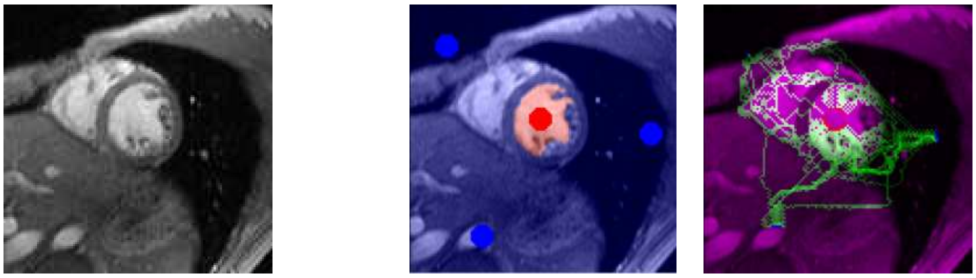
The algorithm terminates if the number of cycles (augmentations) is finite. Since the algorithm is not a shortest path algorithm the polynomial bound for the number of augmentations does not seem to be valid. There is only a trivial bound connected with the value of a minimum cut in case when all costs are integers.

It remains to show that when the algorithm terminates it generates a maximum flow. In fact, both partitions $S, \mathcal{V}-S$ and $\mathcal{V}-T, S$ at the end of the algorithm give minimum s-t-cuts (these cuts are the same if $S \cup T = \mathcal{V}$). Suppose the algorithm has terminated. It could only have happened in the growth stage when no active nodes were left. S and T are disjoint sets such that $s \in S$ and $t \in T$. In order to show that $S, \mathcal{V} - S$ is a minimum cut (and, thus, the current flow is a maximum flow), we need to show that all edges leaving S are saturated. Let (p, q) be such an edge: $TREE(p) = S, TREE(q) \neq S$. Since no active nodes are left then p is passive. Hence, invariant I4 implies that (p, q) is saturated, as desired. The same argument shows that $\mathcal{V} - T, S$ is a minimum cut as well.

5.2.4 Algorithm tuning

Our description leaves many free choices in implementing certain details. For example, we found that the order of processing active nodes and orphans may have a significant effect on the algorithm's running time.

A natural choice for processing active nodes would be a FIFO queue ("First-In-First-Out"). We used a modified version of this strategy, which we found to have a slightly better performance. Namely, we maintain two queues Q_1 and Q_2 which are empty in the beginning. When a node becomes active, it is added to the rear of the second queue Q_2. When we need to retrieve an active node, we first check whether the first queue Q_1 is empty; if it is the case, then we "swap" the two queues. After that we take the node from the front of the first queue Q_1. This strategy guarantees that at least the first path from the source to the sink is



Original image (a) Graph cut (b) Graph flow

Figure 5.2: Graph cut/flow example in the context of image segmentation in section 5.3.4. The cost of edges between the pixels is low in places with high intensity contrast. Red and blue seeds are linked to the source s and the sink t, correspondingly. (a) shows the optimal segmentation (minimum cut) and (b) displays augmenting paths (flow) from s to t. As the flow increases it quickly saturates weak edges that are at the boundary of segments in (a).

the shortest. Note that the search tree may change unpredictably during adoption stage. Thus, we can not guarantee anything about paths found after the first one.

For processing orphans we used the following approach. Let $O \cap S = \{p_1, \ldots, p_n\}$ be the set of orphans in S right after the augmentation stage ordered along the augmenting path (so p_1 was the closest node to the source and p_n the farthest). First we process the node p_1. It may cause its descendants to become orphans; we process them as well using the FIFO order. After all descendants of p_1 are processed, we proceed to the next node p_2, and so on. The order of processing orphans in the set T is similar, except that they ordered according to the distance to the sink.

There are additional free choices in implementing the adoption stage. For example, when an orphan looks for a new parent it may find several qualifying parents. Our description does not say which parent we should pick. One choice would be to pick the first valid parent. We chose a different strategy which helps to get shorter paths: we check all neighbors of the orphans and pick a qualifying parent which has the smallest distance to the terminal (the source or the sink). Note that in order to compute this distance, we need to trace the path from a potential parent to the terminal. However, we need to do this in any case since we must check the origin of a node to determine whether it can be a valid parent.

It may easily happen in the adoption stage that we check the origin of the same node several times. To speed up the algorithm, we use a "marking" heuristic: if a node is confirmed to be connected to a terminal, then we mark it by recording the distance to a terminal and the "timestamp" showing when this distance was computed. In this case other orphans do not have to trace the roots of their potential parents all the way to a terminal. More details about this heuristic are given in the next section.

Finally, during the growth stage we have an opportunity to reassign parents of some nodes in the search trees. Indeed, suppose that an active node p checks its neighbor q and discovers that the edge $tree\_cap(p \to q)$ is non-saturated and q is in the same search tree as p. Then p can be made to be the new parent of q without violating invariants I1-I4, as long as q is not one of grand-parents of p. This will help to speed the algorithm if it results in shorter paths, i.e. if p is closer to the terminal than the current parent of q. We use the marking information to decide whether or not to reassign the parent of q. Details are contained in the next section.

We used a fixed tuning of our algorithm in all experiments of section 5.3 and a library with our implementation is available upon request for research purposes.

Note that augmenting paths on graphs in vision can be easily visualized. In the majority of cases such graphs are regular grids of nodes that correspond to image pixels. Then, augmenting paths and the whole graph flow can be meaningfully displayed (see Figure 5.2). We can also display the search tree at different stages. This allows a very intuitive way of tuning max-flow methods in vision.

5.2.5 "Marking" heuristic

We now present the "marking" heuristic that we used. To implement this heuristic, we store some additional information for each node p in sets S and T:

- $DIST(p)$ - an estimate on the distance to a terminal
- $TS(p)$ - a timestamp showing when this estimate was made

In addition, we maintain a global counter $TIME$ which is incremented in the beginning of each augmentation. In general, $DIST(p)$ shows the correct distance only at the stage when it was computed (i.e. when $TS(p) = TIME$), and at later stages this information becomes outdated. However, we show that some invariants hold at later stages as well, which helps us to ensure algorithm's correctness. These invariants are formulated in the next section.

A modified algorithm with marking taken into account can be formulated as follows. Changes are marked by the star (*).

```
initialize:  S = {s}, T = {t}, A = {s,t}, O = ∅
(*)            DIST(s) = DIST(t) = 0
(*)            TS(s) = TS(t) = TIME = 1
while true
           grow S or T to find an augmenting path P from s to t
           if P = ∅ terminate
(*)        TIME := TIME + 1
           augment on P
           adopt orphans
end while
```

Modified versions of *growth* and *adoption* stages are described below (the *augmentation* stage is the same as before).

Growth stage:

There are two changes to the growth stage. First, when an active node acquires a new child from the set of free nodes, an additional operation is performed: the child's distance to a terminal and its timestamp are set to be the parent's distance (incremented by one) and the parent's timestamp, respectively. Second, when an active node encounters another node in the same search tree, it checks whether the parent of the latter node can be reassigned.

```
while A ≠ ∅
   pick an active node p ∈ A
   for every neighbor q such that tree_cap(p → q) > 0
      if TREE(q) = ∅ then add q to search tree as an active node:
           TREE(q) := TREE(p), PARENT(q) := p, A := A ∪ {q}
(*)        DIST(q) := DIST(p) + 1, TS(q) := TS(p)
(*)   if TREE(q) = TREE(p) and isCloser(p, q)
(*)   then reassign the parent of q to be p:
(*)        PARENT(q) := p
(*)        DIST(q) := DIST(p) + 1, TS(q) := TS(p)
      if TREE(q) ≠ ∅ and TREE(q) ≠ TREE(p) return P = PATH_{s→t}
   end for
   remove p from A
end while
return P = ∅
```

$isCloser(p, q)$ is a predicate which tries to determine whether p is closer to a terminal than q. It is defined as follows:

$$isCloser(p, q) = TS(q) \leq TS(p) \text{ and } DIST(q) > DIST(p)$$

The first condition is necessary to ensure the algorithm's correctness: it will guarantee that q is not a grand-parent of p (see next section). Without this condition cycles could be formed.

Adoption stage:

A general structure of the adoption stage remains the same:

```
while O ≠ ∅
    pick an orphan node p ∈ O and remove it from O
    process p
end while
```

However, details of the operation "`process` p" are slightly different.

First we are trying to find a new valid parent for p among its neighbors. As before, a valid parent q should satisfy: $TREE(q) = TREE(p)$, $tree\_cap(q \to p) > 0$, and the "origin" of q should be either source or sink. Checking the origin of q can now be described as follows. We start to trace the path from q to a terminal checking the condition $TS(r) = TIME$ for nodes r along the path. If for some node r this condition is true, then it means that the origin of r was checked during the same iteration and $DIST(r)$ shows the correct distance to a terminal (see next section for a proof of this fact). Thus, we can stop there and use $DIST(r)$ for computing the distance from q to a terminal. If we found that the origin of q is a terminal, then we perform an additional operation: we trace the same path again, assigning the correct distance information to nodes r in the path and setting $TS(r) = TIME$.

If node p finds a new valid parent q then we set $PARENT(p) = q$, $DIST(p) = DIST(q) + 1$, $TS(p) = TIME$. In this case p remains in its search tree and the active (or passive) status of p remains unchanged. If p does not find a valid parent then we perform the same operations as before:

- scan all neighbors q of p such that $TREE(q) = TREE(p)$:

  – if $tree\_cap(q \to p) > 0$ add q to the active set A

  – if $PARENT(q) = p$ add q to the set of orphans O and set $PARENT(q) := \emptyset$

- $TREE(p) := \emptyset,\ \ A := A - \{p\}$

5.2.6 Correctness of the marking heuristic

In order to prove correctness of the modified algorithm, we need to introduce additional invariants which are maintained during the execution of the algorithm.

I5 *For all nodes p in the search forests S and T $TS(p) \leq TIME$.*

I6 *If a node p is in the search forest S or T and $TS(p) = TIME$ then the origin of p is a terminal and $DIST(p)$ shows the correct distance to this terminal.*

I7 *If a node p is in the search forest S or T and q is a child of p then $TS(q) \leq TS(p)$.*

I8 *If a node p is in the search forest S or T, q is a child of p, and $TS(q) = TS(p)$ then $DIST(q) > DIST(p)$.*

Invariants I1-I8 are clearly true at the initialization of the algorithm. It is straightforward to check that each operation maintains these invariants. We will discuss only potential pitfall. We will show that if $isCloser(p, q)$ is true for nodes p, q in the same search tree S or T, then q cannot be a predecessor of p. This will ensure that cycles are not formed in the growth stage.

Suppose that q is a predecessor of p. Let $r_0 = q, \ldots, r_k = p$ be the path from q to p ($k \geq 1$). Invariant I7 says that

$$TS(q) = TS(r_0) \geq TS(r_1) \geq \ldots \geq TS(r_k) = TS(p)$$

Since $isCloser(p, q)$ is true, then $TS(q) \leq TS(p)$. Therefore,

$$TS(q) = TS(r_0) = TS(r_1) = \ldots = TS(r_k) = TS(p)$$

Thus, according to invariant I8

$$DIST(q) = DIST(r_0) < TS(r_1) < \ldots < DIST(r_k) = DIST(p)$$

which contradicts to the condition $isCloser(p, q)$.

5.3 Experimental tests on applications in vision

In this section we experimentally test min-cut/max-flow algorithms for three different applications in computer vision: image restoration (section 5.3.2), stereo (section 5.3.3), and object segmentation (section 5.3.4). We chose formulations where certain appropriate versions of energy (1.1) can be minimized via graph cuts. The corresponding graph structures were previously described by [GPS89, IG98b, BVZ98, BVZ01, KZ01, KZ02a, BJ01] in detail. These (or very similar) structures are used in all computer vision applications with graph cuts (that we are aware of) to date.

5.3.1 Experimental setup

Note that we could not test all known min-cut/max-flow algorithms. In our experimental tests on graph-based energy minimization methods in vision we compared the new algorithm in section 5.2 and the following standard min-cut/max-flow algorithms outlined in section 5.1.1:

DINIC: Dinic's algorithm [Din70].

H_PRF: Push-Relabel algorithm [GT88] with the highest level selection rule.

Q_PRF: Push-Relabel algorithm [GT88] with the queue based selection rule.

Many previous experimental tests, including the results in [CG97], show that the last two algorithms work consistently better than a large number of other min-cut/max-flow algorithms of combinatorial optimization. The theoretical worst case complexities for these "push-relabel" algorithms are $O(n^3)$ for Q_PRF and $O(n^2\sqrt{m})$ for H_PRF.

For DINIC, H_PRF, and Q_PRF we used the implementations written by Cherkassky and Goldberg [CG97], except that we converted them from C to C++ style and modified the interface (i.e. functions for creating a graph). Both H_PRF and Q_PRF use global and gap relabeling heuristics.

Our algorithm was implemented in C++. We selected a tuning described in section 5.2.4. We did not make any machine specific optimization (such as pipeline-friendly instruction scheduling or cache-friendly memory usage).

Experiments in sections 5.3.2 and 5.3.4 were performed on 1.4GHz Pentium IV PC (2GB RAM, 8KB L1 cache, 256KB L2 cache), and experiments in section 5.3.3 were performed on UltraSPARC II workstation with four 450 MHz processors and 4GB RAM. In the former case we used Microsoft Visual C++ 6.0 compiler, Windows NT platform and in the latter case - GNU C++ compiler, version 3.2.2 with the flag "-O5", SunOS 5.8 platform. To get system time, we used ftime() function in Unix and _ftime() function in Windows. Although these functions do

(a) *Diamond* restoration (b) Original *Bell Quad* (c) "Restored" *Bell Quad*

Figure 5.3: Image Restoration Examples

not measure process computation time, we felt that they were appropriate since we got very consistent results (within 1%) when running tests multiple times.

5.3.2 Image restoration

Image restoration is a representative early vision problem. The goal is to restore original pixel intensities from the observed noisy data. Some examples of image restoration are shown in Figure 5.3. The problem can be very easily formulated in terms of energy (1.1) minimization. In fact, many other low level vision problems can be represented by the same energies. We chose the context of image restoration mainly for its simplicity.

In this section we consider two examples of energy (1.1) based on the Potts and linear models of interaction, correspondingly. Besides image restoration [GPS89], graph methods for minimizing Potts energy were used in segmentation [KFT$^+$00], stereo [BVZ98, BVZ01, KZ02a], object recognition [BH99], voxel occupancy [SVZ00], and augmented reality [TBRN00]. Linear interaction energies were used in stereo [RC98] and segmentation [IG98b]. Minimization of the linear interaction energy is based on graphs that are quite different from what is used for the Potts model. At the same time, there is very little variation between the graphs in different applications when the same type of energy is used. They mainly differ in their specific edge cost settings while the topological properties of graphs are almost identical once the energy model is fixed.

Potts Model

The Potts energy that we use for image restoration is

$$E(I) = \sum_{p\in\mathcal{P}} |I_p - I_p^o| + \sum_{\{p,q\}\in\mathcal{N}} K_{\{p,q\}} \cdot\ T(I_p \neq I_q) \tag{5.1}$$

where $I = \{I_p\ |p \in \mathcal{P}\}$ is a vector of unknown "true" intensities of pixels on the image $\mathcal{P}$ and $I^o = \{I_p^o\ |p \in \mathcal{P}\}$ are intensities observed in the original image corrupted by noise. The Potts interactions are specified by penalties $K_{\{p,q\}}$ for intensity discontinuities between pairs of neighboring pixels. Function $T(\cdot)$ is 1 if the condition inside parenthesis is true and 0 otherwise. In the case of two labels the Potts energy can be minimized exactly using the graph cut method of Greig et al. [GPS89].

We consider image restoration with multiple labels where the problem becomes NP-hard. We use iterative α-*expansion* method in [BVZ01] which is guaranteed to find a solution within a factor of two from the global minimum of the Potts energy. Details of this algorithms are described in section 2.4.

Two tables below present the running times (in seconds, 1.4GHz Pentium IV) when different max-flow/min-cut algorithms are employed in the basic step of each α-expansion. Each table corresponds to one of the original images shown in Figure 5.3. The number of allowed labels is 210 (*Diamond*) and 244 (*Bell Quad*), correspondingly. We run the algorithms on images at different resolutions. At each column we state the exact size (HxW) in pixels. Note that the total number of pixels increases by a factor of 2 from left to right.

method	**input:** Diamond, 210 labels						
	35x35	50x50	70x70	100x100	141x141	200x200	282x282
DINIC	0.39	0.77	3.42	4.19	13.85	43.00	136.76
H_PRF	0.17	0.34	1.16	1.68	4.69	12.97	32.74
Q_PRF	0.16	0.35	1.24	1.70	5.14	14.09	40.83
Ours	0.16	0.20	0.71	0.74	2.21	4.49	12.14

method	**input:** Bell Quad, 244 labels					
	44x44	62x62	87x87	125x125	176x176	250x250
DINIC	1.32	4.97	13.49	37.81	101.39	259.19
H_PRF	0.31	0.72	1.72	3.85	8.24	18.69
Q_PRF	0.20	1.00	1.70	4.31	10.65	25.04
Ours	0.19	0.48	0.98	2.11	4.84	10.47

Note that the running times above correspond to the end of the first cycle of the α-expansion method in [BVZ01] when all labels were expanded once. The relative speeds of different max-flow/min-cut algorithms do not change much when the energy minimization is run to convergence. The number of cycles it takes to converge can vary from 1 to 3 for different resolutions/images. Thus, the running times to convergence are hard to compare between the columns and we do not present them. In fact, restoration results are quite good even after the first iteration. In most cases additional iterations do not improve the actual output much. Figure 5.3(a) shows the result of the Potts model restoration of the *Diamond* image (100x100) after the first cycle of iterations.

Linear interaction energy

Here we consider image restoration with "linear" interaction energy. Figure 5.3(c) shows one restoration result that we obtained in our experiments with this energy. The linear interaction energy can be written as

$$E(I) = \sum_{p\in\mathcal{P}} |I_p - I^o_p| + \sum_{\{p,q\}\in\mathcal{N}} A_{\{p,q\}} \cdot \ |I_p - I_q| \tag{5.2}$$

where constants $A_{\{p,q\}}$ describe the relative importance of interactions between neighboring pixels p and q. If the set of labels is finite and ordered then this energy can be minimized exactly using either of the two almost identical graph-based methods developed in [IG98b, BVZ98]. In fact, both of these methods use graphs that are very similar to the one introduced by [RC98] in the context of multi-camera stereo. The graphs are constructed by consecutively connecting multiple layers of image-grids. Each layer corresponds to one label. The two terminals are connected only to the first and the last layers. Note that the topological structure of these graphs is noticeably different from the Potts model graphs especially when the number of labels (layers) is large.

The tables below show the running times (in seconds on 1.4GHz, Pentium IV) that different min-cut/max-flow algorithms took to compute the exact minimum of the linear interactions energy (5.2). We used the same *Diamond* and *Bell Quad* images as in the Potts energy tests. We run the algorithms on images at different resolution. At each column we state the exact size (hight and width) in pixels. Note that the total number of pixels increases by a factor of 2 from left to right.

method	**input:** Diamond, 54 labels					
	35x35	50x50	70x70	100x100	141x141	200x200
DINIC	1.34	4.13	8.86	18.25	34.84	57.09
H_PRF	0.47	1.30	3.03	7.48	17.53	43.58
Q_PRF	0.55	1.16	3.05	6.50	12.77	22.48
Ours	0.17	0.33	0.63	1.41	2.88	5.98

method	**input:** Bell Quad, 32 labels					
	44x44	62x62	87x87	125x125	176x176	250x250
DINIC	0.55	1.25	2.77	6.89	15.69	31.91
H_PRF	0.48	1.25	2.75	7.42	17.69	38.81
Q_PRF	0.27	0.56	1.55	2.39	6.78	10.36
Ours	0.13	0.27	0.52	1.09	2.33	4.84

The structure of linear interaction graph directly depends on the number of labels[2]. In fact, if there are only two labels then the graph is identical to the Potts model graph. However, both size and topological properties of the linear interaction graphs change as the number of labels (layers) gets larger and larger. Below we compare the running times of the algorithms for various numbers of allowed labels (layers). We consider the same two images, *Diamond* and *Bell Quad*. In each case the size of the corresponding image is fixed. At each column we state the number of allowed labels $\mathcal{L}$. The number of labels increases by a factor of 2 from left to right.

[2]Note that in section 5.3.2 we tested multi-label Potts energy minimization algorithm [BVZ01] where the number of labels affects the number of iterations but has no effect on the graph structures.

method	**input:** Diamond, 100x100 (pix)			
	$\mathcal{L}$=27	$\mathcal{L}$=54	$\mathcal{L}$=108	$\mathcal{L}$=215
DINIC	6.89	18.16	50.81	166.27
H_PRF	3.05	7.38	15.50	47.49
Q_PRF	2.36	6.41	17.22	43.47
Ours	0.55	1.39	4.34	16.81

method	**input:** Bell Quad, 125x125 (pix)			
	$\mathcal{L}$=32	$\mathcal{L}$=63	$\mathcal{L}$=125	$\mathcal{L}$=250
DINIC	6.91	17.69	46.64	102.74
H_PRF	7.47	19.30	58.14	192.39
Q_PRF	2.39	7.95	15.83	45.64
Ours	1.13	2.95	10.44	41.11

Our experiments with linear interaction graphs show that most of the tested max-flow/min-cut algorithms are close to linear with respect to increase in image size. At the same time, none of the algorithms behaved linearly with respect to the number of labels despite the fact that the size of graphs linearly depend on the number of labels. Our algorithm is a winner in absolute speed as in most of the tests it is 2-4 times faster than the second best method. However, our algorithm's dynamics with respect to increase in the number of labels is not favorable. For example, Q_PRF gets real close to the speed of our method in case of $\mathcal{L}$=250 (*Bell Quad*) even though our algorithm was 2 times faster than Q_PRF when the number of labels was $\mathcal{L}$=32.

5.3.3 Stereo

Stereo and its generalization to multiple cameras are other classical vision problems where graph-based energy minimization methods have been successfully applied. We consider three graph-based methods for solving this problem: pixel-labeling stereo with the Potts model [BVZ01], algorithm I and algorithm II (chapter 4). We applied the method of [BVZ01] and algorithm II to the case of two cameras, and algorithm I to the case of multiple cameras.

Pixel-labeling stereo with the Potts model

First, we consider a formulation of stereo problem given in [BVZ98, BVZ01] (see section 2.4). The tests were done on three stereo examples. We used the *Head* pair from the University of Tsukuba, and the well-known *Tree* pair from SRI (see section 4.5). To diversify our tests we compared the speed of algorithms on a *Random* pair where the left and the right images did not correspond to the same scene (they were taken from the *Tsukuba* and the *Tree* pairs, respectively).

Running times for these stereo examples are shown in seconds (450 MHz Ultra-SPARC II Processor) in the table below. As in the restoration section, the running times correspond to the first cycle of the algorithm. The relative performance of different max-flow/min-cut algorithms is very similar when the energy minimization is run to convergence while the number of cycles it takes to converge varies between 3 and 5 for different data sets. We performed two sets of experiments: one with a four-neighborhood system and the other with an eight-neighborhood system. The corresponding running times are marked by "N4" and "N8". The disparity maps in N4 and N8 cases were very similar.

method	*Tsukuba*, 384x288		*Tree*, 256x233		*Random*, 384x288	
	N4	N8	N4	N8	N4	N8
DINIC	104.18	151.32	9.53	19.80	105.93	167.16
H_PRF	12.00	18.03	1.65	2.86	14.25	18.22
Q_PRF	10.40	14.69	2.13	3.33	12.05	15.64
Ours	3.41	6.47	0.68	1.42	3.50	6.87

Algorithm II

We used the same three datasets as in the previous section. Running times for these stereo examples are shown in seconds (450 MHz UltraSPARC II Processor) in the table below. The times are for the first cycle of the algorithm.

method	*Tsukuba*, 384x288		*Tree*, 256x233		*Random*, 384x288	
	N4	N8	N4	N8	N4	N8
DINIC	376.70	370.94	66.19	102.60	81.70	115.58
H_PRF	35.65	49.81	9.07	15.41	5.48	8.32
Q_PRF	33.12	44.86	8.55	13.64	9.36	14.02
Ours	10.64	19.14	2.73	5.51	3.61	6.42

Multi-camera stereo: algorithm I

The tests were done for three datasets: the *Tsukuba* sequence from the University of Tsukuba, the *Garden* sequence and the *Dayton* sequence (see section 4.5). The table below gives running times (in seconds, 450 MHz UltraSPARC II Processor) for these three datasets. The times are for the first cycle of the algorithm.

method	input sequence		
	Tsukuba 5 views 384x288	*Garden* 8 views 352x240	*Dayton* 5 views 384x256
DINIC	2793.48	2894.85	2680.91
H_PRF	282.35	308.52	349.60
Q_PRF	292.93	296.48	266.08
Our	104.33	81.30	85.56

5.3.4 Segmentation

In this section we describe experimental tests that compare running times of the selected min-cut/max-flow algorithms for segmentation technique in [BJ01]. The graph-based method in [BJ01] allows to segment an object of interest in N-D images/volumes. This technique generalizes the MAP-MRF method of Greig at. al. [GPS89] by incorporating additional hard constraints into minimization of the Potts energy

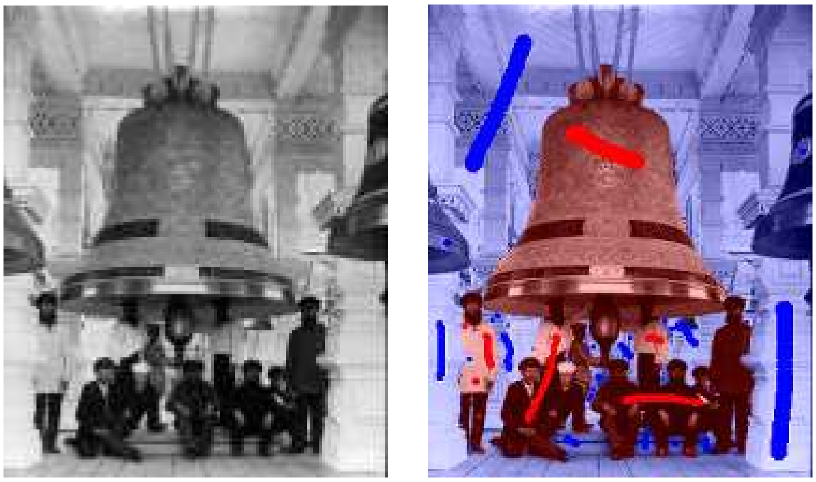
$$E(L) = \sum_{p\in\mathcal{P}} D_p(L_p) + \sum_{\{p,q\}\in\mathcal{N}} K_{\{p,q\}} \cdot \; T(L_p \neq L_q)$$

over binary (object/background) labelings. The hard constraints may come from a user placing object and background seeds.

The technique finds a globally optimal binary segmentation of N-dimensional image satisfying the hard constraints (seeds). The computation is done in one pass of a max-flow/min-cut algorithm on a certain graph. In case of 2D images the structure of the underlying graph is exactly the same as shown in Figure 2.3. In 3D cases [BJ01] build a regular 3D grid graph.

We tested min-cut/max-flow algorithms on 2D and 3D segmentation examples illustrated in Figure 5.4. This figure demonstrates original data and our segmen-

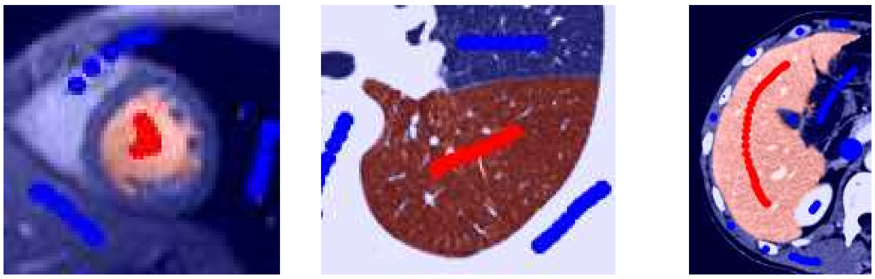
Photo Editing



(a) Bell Photo (b) Bell Segmentation

Medical Data

(c) Cardiac MR (e) Lung CT (g) Liver MR



(d) LV Segment (f) Lobe Segment (h) Liver Segment

Figure 5.4: Segmentation Experiments

tation results corresponding to some sets of seeds. Note that the user can place seeds interactively. New seeds can be added to correct segmentation imperfections. The technique in [BJ01] efficiently recomputes the optimal solution starting at the previous segmentation result.

Figure 5.4(a-b) shows one of our experiments where a group of people around a bell were segmented on a real photo image (255x313 pixels). Other segmentation examples in (c-h) are for 2D and 3D medical data. In (c-d) we segmented a left ventricle in a 3D cardiac MR data (127x127x12 voxels). In our 3D experiments the seeds were placed in one slice in the middle of the volume. Often, this is enough to segment the whole volume correctly. The tests with lung CT data (e-f) were made in 2D (409x314 pixels) case. The goal was to segment out a lower lung lobe. In (g-h) we tested the algorithms on 2D liver MR data (511x511 pixels). Additional 3D experiments were performed on heart ultrasound and kidney MR volumes.

The tables below compare running times (in seconds, 1.4GHz Pentium IV) of the selected min-cut/max-flow algorithms for a number of segmentation examples. Note that these times include only min-cut/max-flow computation[3]. In each column we show running times of max-flow/min-cut algorithms corresponding to exactly the same set of seeds. The running times were obtained for the "6" and "26" neighborhood systems (N6 and N26). Switching from N6 to N26 increases complexity of graphs but does not affect the quality of segmentation results much.

[3]Time for entering seeds may vary between different users. For the experiments in Figure 5.4 all seeds were placed within 10 to 20 seconds.

method	2D examples					
	Bell photo (255x313)		Lung CT (409x314)		Liver MR (511x511)	
	N4	N8	N4	N8	N4	N8
DINIC	2.73	3.99	2.91	3.45	6.33	22.86
H_PRF	1.27	1.86	1.00	1.22	1.94	2.59
Q_PRF	1.34	0.83	1.17	0.77	1.72	3.45
Ours	0.09	0.17	0.22	0.33	0.20	0.45

method	3D examples					
	Heart MR (127x127x12)		Heart US (76x339x38)		Kidney MR (99x66x31)	
	N6	N26	N6	N26	N6	N26
DINIC	20.16	39.13	172.41	443.88	3.39	8.20
H_PRF	1.38	2.44	18.19	47.99	0.19	0.50
Q_PRF	1.30	3.52	23.03	45.08	0.19	0.53
Ours	0.70	2.44	13.22	90.64	0.20	0.58

5.4 Conclusions

We have experimented with a reasonable sample of typical vision graphs. In most examples our new min-cut/max-flow algorithm worked 2-5 times faster than any of the other methods, including the push-relabel and Dinic's algorithms (which are known to outperform other min-cut/max-flow techniques). In some cases the new algorithm made possible near real-time performance of the corresponding applications.

More specifically, we can conclude that our algorithm is consistently several times faster than the second best method in all applications where the graphs are 2D grids. However, our algorithm's performance degrades when the complexity of the underlying graphs is increased. For example, on linear interaction energy graphs (section 5.3.2) with a large number of grid-layers (labels), the performance of Q_PRF was comparable to our algorithm. Similarly, experiments in section 5.3.4 show that push-relabel methods (H_PRF and Q_PRF) are comparable to our algorithm in 3D segmentation tests even though it was several times faster in all 2D segmentation examples. Going from the 6-connected neighborhood system to the 26-connected system further decreased the relative performance of our method in 3D segmentation.

Note that we do not have a polynomial bound for our algorithm.[4] Interestingly, in all our tests on 2D and 3D graphs that occur in real computer vision applications, our algorithm significantly outperformed a polynomial method of DINIC. Our results suggest that grid graphs in vision are a very specific application for min-cut/max-flow algorithms. In fact, Q_PRF outperformed H_PRF in many of our tests (especially in section 5.3.2) despite the fact that H_PRF is generally regarded as the fastest algorithm in combinatorial optimization community.

[4]The trivial bound given in section 5.2 involves the cost of a minimum cut and, theoretically, it is not a polynomial bound. In fact, additional experiments showed that our algorithm is by several orders of magnitude slower than Q_PRF, H_PRF, and DINIC on several standard (not from computer vision) types of graphs commonly used for tests in combinatorial optimization community.

Chapter 6

Conclusions

This thesis has explored graph cut techniques in computer vision. In this chapter we summarize our contributions and state some open problems.

6.1 Contributions

We have showed how to apply graph cut techniques to the problem of scene reconstruction from multiple views. The algorithms that we developed have several important properties: they treat the input images symmetrically, handle visibility properly, and impose spatial smoothness while also preserving discontinuities. Experimental results show that our algorithms for two views are among the top performers, and that the accuracy of the reconstruction is greatly improved when more views are used.

We also addressed two issues common to many other vision algorithms that use graph cuts. First, we gave a partial characterization of the class of functions of binary variables that can be minimized via graph cuts. We showed that regularity is a necessary condition for graph representability, and that is sufficient for the class $\mathcal{F}^3$. In addition, we gave a general-purpose graph construction to minimize any regular function in $\mathcal{F}^3$. It is thus no longer to necessary to construct graphs explicitly for minimizing a function; instead it suffices to show that the function is regular. Second, we presented a new maxflow algorithm for computing minimum s-t cuts that outperforms other standard algorithms for many vision applications, as we demonstrated experimentally. It speeds up many graph-based methods, including the scene reconstruction algorithms presented in this thesis.

6.2 Open questions

In conclusion we mention some open questions that we would like to explore in the future.

Minimizing a broader class of energy functions

- We proved that regularity is a necessary condition for graph representability for any function of binary variables. However, we showed that it is also sufficient only for class $\mathcal{F}^3$. An obvious question is whether regular functions in other classes ($\mathcal{F}^4$, $\mathcal{F}^5$, $\ldots$) are graph-representable. Note that a positive answer would give practical algorithms only for classes $\mathcal{F}^k$ with small k since the size of the graph would almost certainly be exponential in k.

- We gave a condition only for functions of binary variables. Can we say something useful about functions of multi-valued variables? Of course, we could choose an encoding of multi-valued variables with binary variables and then apply our results. However, it would be much more powerful to have a condition that does not depend on this encoding.

Multi-camera scene reconstruction

- As is common in stereo, we assumed that surfaces are Lambertian. Unfortunately, in real scenes this assumption often fails. Can we somehow relax it?

- Currently we have a restriction on the geometry of cameras and the object, which prevents, for example, the cameras from surrounding the object. Can we modify our algorithm to more general geometries?

- Ideally, the smoothness term should impose a constraint on the shape of the object, and therefore should not depend on position of the cameras. However, both smoothness terms that we use are formulated in terms of cameras viewing the object. Can we reformulate the scene reconstruction problem to overcome this drawback?

- As is common with energy minimization algorithms, we pick the parameters of our energy function heuristically. Is there a theoretically sound model that would explain how to choose these parameters?

Despite these remaining open questions, it is clear that graph cut algorithms can now be used to obtain strong results for multi-camera scene reconstruction.

BIBLIOGRAPHY

[Bar89] Stephen Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, 1989.

[BGCM02] C. Buehler, S. J. Gortler, M. Cohen, and L. McMmillan. Minimal surfaces for stereo vision. In *European Conference on Computer Vision*, volume 3, pages 885–899, 2002.

[BH99] Yuri Boykov and Daniel Huttenlocher. A new bayesian framework for object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 517–523, 1999.

[BJ01] Yuri Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *International Conference on Computer Vision*, pages I: 105–112, 2001.

[BK01] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in computer vision. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 2134 of *LNCS*, pages 359–374. Springer-Verlag, September 2001.

[BK03] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *International Conference on Computer Vision*, 2003.

[BT98] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401–406, April 1998.

[BT99] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. In *International Conference on Computer Vision*, pages 489–495, 1999.

[BVZ98] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov Random Fields with efficient approximations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–655, 1998.

[BVZ99] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, pages 377–384, September 1999.

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

[BZ87] A. Blake and A. Zisserman. *Visual Reconstruction.* MIT Press, 1987.

[CB92] R. Cipolla and A. Blake. Surface shape from the deformation of apparent contours. *International Journal of Computer Vision*, 9(2):83–112, November 1992.

[CCPS98] William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1998.

[CG97] B. V. Cherkassky and A. V. Goldberg. On implementing push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.

[Cun85] W. H. Cunningham. Minimum cuts, modular functions, and matroid polyhedra. *Networks*, 15:205–215, 1985.

[DE87] H. Derin and H. Elliot. Modeling and segmentation of noisy and textured images using Gibbs Random Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(1):39–55, 1987.

[Din70] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[DL02] Jose Dias and Jose Leitao. The $\mathcal{Z}\pi M$ algorithm: A method for interferometric image reconstruction in SAR/SAS. *IEEE Transactions on Image Processing*, 11(4):408–422, April 2002.

[FF62] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[Fuj90] S. Fujishige. *Submodular functions and Optimization*, volume 47 of *Annals of Discrete Mathematics*. North Holland, 1990.

[GG84] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[GM03] B Goldlucke and M.A. Magnor. Joing 3D-reconstruction and background separation in multiple views using graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 683–688, 2003.

[GPS89] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

[GT88] A. Goldberg and R. Tarjan. A new approach to the maximum flow problem. *Journal of the Association for Computing Machinery*, 35(4):921–940, October 1988.

[HPB98] T. Hofmann, J. Puzicha, and J. M. Buhmann. Unsupervised texture segmentation in a deterministic annealing framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):803–818, 1998.

[HS81] B. K. P. Horn and B. Schunk. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[IFF00] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. In *ACM Symposium on Theoretical Computer Science*, pages 97–106, 2000.

[IG98a] H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *European Conference on Computer Vision*, pages 232–248, 1998.

[IG98b] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 125–131, 1998.

[Ish03] Hiroshi Ishikawa. Exact optimization for Markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003. To appear.

[Kar99] David R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, May 1999.

[KFT+00] Junmo Kim, John Fisher, Andy Tsai, Cindy Wible, Alan Willsky, and William Wells. Incorporating spatial priors into an information theoretic approach for fMRI data analysis. In *Medical Image Computing and Computer-Assisted Intervention*, pages 62–71, 2000.

[KS00] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):197–216, July 2000.

[KSC01] S.B. Kang, R. Szeliski, and J. Chai. Handling occlusions in dense multiview stereo. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001. Expanded version available as MSR-TR-2001-80.

[KSE+03] Vivek Kwatra, Arno Schoedl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of ACM SIGGRAPH*, 2003.

[KZ01] Vladimir Kolmogorov and Ramin Zabih. Visual correspondence with occlusions using graph cuts. In *International Conference on Computer Vision*, pages 508–515, 2001.

[KZ02a] Vladimir Kolmogorov and Ramin Zabih. Multi-camera scene reconstruction via graph cuts. In *European Conference on Computer Vision*, volume 3, pages 82–96, 2002.

[KZ02b] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? In *European Conference on Computer Vision*, volume 3, pages 65–81, 2002. Expanded version available as Cornell CS technical report CUCS-TR2001-1857.

[KZ03] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003. To appear.

[KZG03] Vladimir Kolmogorov, Ramin Zabih, and Steven Gortler. Generalized multi-camera scene reconstruction using graph cuts. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, July 2003.

[Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, February 1994.

[Li95] S. Li. *Markov Random Field Modeling in Computer Vision*. Springer-Verlag, 1995.

[LT03] M.H. Lin and C. Tomasi. Surfaces with occlusions from layered stereo. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 710–717, 2003.

[MA83] W.N. Martin and J.K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158, March 1983.

[MP76] D. Marr and T.A. Poggio. Cooperative computation of stereo disparity. *Science*, 194(4262):283–287, October 1976.

[PTK85] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.

[RC98] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *International Conference on Computer Vision*, 1998.

[Sch00] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory*, B80:346–355, 2000.

[SD99] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):1–23, November 1999.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, April 2002.

[SVZ00] Dan Snow, Paul Viola, and Ramin Zabih. Exact voxel occupancy with graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 345–352, 2000.

[SZ99] Richard Szeliski and Ramin Zabih. An experimental comparison of stereo algorithms. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, number 1883 in LNCS, pages 1–19, Corfu, Greece, September 1999. Springer-Verlag.

[Sze93] R. Szeliski. Rapid octree construction from image sequences. *Computer Vision, Graphics and Image Processing*, 58(1):23–32, July 1993.

[TBRN00] B. Thirion, B. Bascle, V. Ramesh, and N. Navab. Fusion of color, shading and boundary information for factory pipe segmentation. In *CVPR*, pages 349–356. IEEE, 2000.

[TP86] V. Torre and T. Poggio. On edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(2):147–163, 1986.

[Vek99] Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, August 1999. Available as technical report CUCS-TR-2000-1787.

[Vek00] O. Veksler. Image segmentation by nested cuts. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages I:339–344, 2000.

[Zal02] Boris Zalesky. Fast algorithms of bayesian segmentation of images. *arXiv*, math.PR/0206184, June 2002.