Paul Alois Nigsch BSc

Approximation algorithms for graph cuts

MASTER THESIS

written to obtain the academic degree of a

Master of Science (MSc)

Master's programme Mathematical Computer Science

submitted to

Graz University of Technology

Supervisor:
Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas POCK

Institute for Computer Graphics and Vision

Graz, July 2015

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Datum/Date	Unterschrift/Signature

Approximation algorithms for graph cuts

Paul Alois Nigsch

July 5, 2015

Abstract

Graph cuts play a important role in computer vision. They are used as central building blocks in a lot of applications. For these computer vision applications, often an approximation of the optimal cut is sufficient. Therefore, fast approximation algorithms are desired, since computer vision problems are often time critical.

Recently a new class of approximation algorithms for the max-flow and min-cut problem, utilizing electrical flows, were developed. These algorithms have a superior asymptotic runtime compared to other approximation algorithms. In this thesis two algorithms, utilizing electrical flows, are presented and evaluated on problem instances appearing in computer vision.

Furthermore, graph cut problems can also be solved by primal-dual algorithms for convex optimization problems. Such a primal-dual algorithm is presented and a modification for speeding up its runtime are evaluated.

It is well known that one can obtain the solution of a special class of graph cuts by solving the famous Rudin-Osher-Fatemi (ROF) functional. This functional can be solved by a recently presented and very fast method, called block coordinate descent. This approach is discussed and compared to the primal-dual algorithm and another state of the art graph cut algorithm.

Since graph cut instances in computer vision often have some special structure, the focus of the evaluation lies on such graphs.

Kurzfassung

Graph cuts oder Min-Cut Probleme kommen als zentrale Komponenten in vielen Computer Vision Anwendungen zum Einsatz. Da einerseits oftmals eine Approximation des optimalen Cut ausreichend ist und andererseits diese Anwendungen in den vielen Fällen laufzeitkritisch sind, sind schnelle Approximationsalgorithmen von besonderem Interesse.

In letzter Zeit wurden einige Algorithmen publiziert, welche so genannte elektrische Flüsse verwenden. Diese erlauben die Konstruktion von Algorithmen mit stark verbesserter Laufzeit im Vergleich zu früheren Algorithmen.

In dieser Arbeit werden zwei Algorithmen, welche elektrische Flüsse verwenden, vorgestellt. Weiters werden diese Algorithmen auf ihre praktische Verwendbarkeit, insbesondere im Hinblick auf Probleminstanzen wie sie typischerweise in der Computer Vision auftreten, untersucht.

Durch die Formulierung als konvexes Optimierungsproblem können Min-Cut Probleme auch mittels Primal-dual Algorithmen, wie sie aus der konvexen Optimierung bekannt sind, gelöst werden.

Weiters können Min-Cut Probleme mit speziellen Strukturen mit Hilfe des bekannten Rudin-Osher-Fatemi (ROF) Funktional berechnet werden.

Zur Lösung dieses Funktionals wurden vor kurzem Methoden vorgestellt, welche ‚block coordinate descent' Methoden verwenden. Die Herangehensweise dieser Methode wird wiederum präsentiert und mit den aktuellen State of the Art Algorithmen verglichen.

Der Fokus liegt dabei wieder auf Probleminstanzen wie sie typischerweise in der Computer Vision vorkommen.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Max-flow and min-cut problems are a class of fundamental problems which are encountered frequently in different fields of mathematics. Including theoretical combinatorial problems like bipartite matching problems and graph connectivity problems, to name but a few. Additionally various practical problems can be solved by max-flows and min-cuts, like distributing water, or some more abstract quantity (like traffic for instance), in some kind of network.

Another field where graph cuts are frequently used is computer vision where graph cuts are used in various applications. The first computer vision model using graph cuts was described by Greiget al. [30] in 1985. In their work the authors used graph cuts in order to denoise binary images and the underlying idea is to treat each pixel as a vertex and each neighbourhood pixel corresponds to an adjacent vertex. By this time graph cuts allowed for a fast and exact solution compared to the previously used metaheuristics, like simulated annealing.

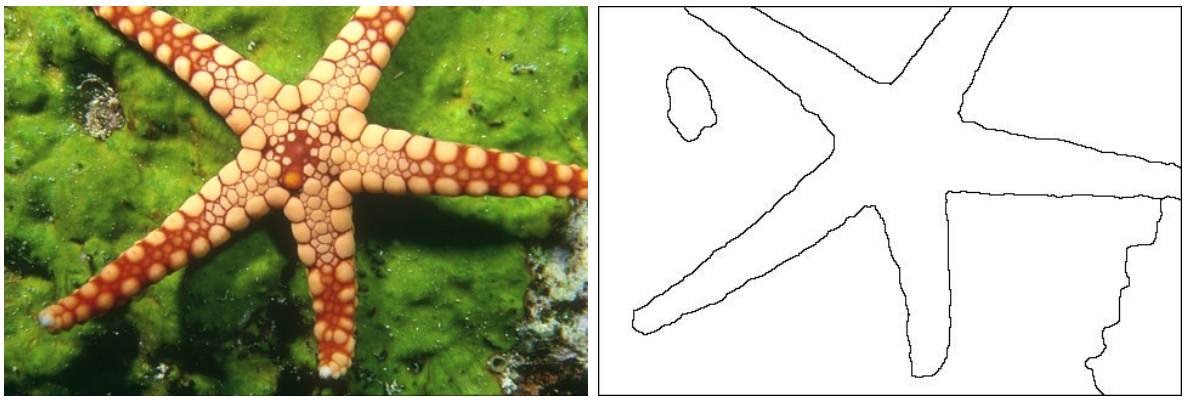
In general, graph cuts can be used whenever some special kind of binary energy minimization has to be performed. Energy functions which can be solved exactly by graph cuts are called 'submodular functions' and this class is very well characterized [37].

Submodular functions arise when one tries to solve Markov random fields. The well know agorithm 'GrabCut' [52] is an example of using this technique.

A third problem in computer vision that can be tackled by graph cuts is the stereo correspondence problem [36, 11]. This problem can also be formulated with Markov random fields and therefore again benefits from fast min-cut algorithms.

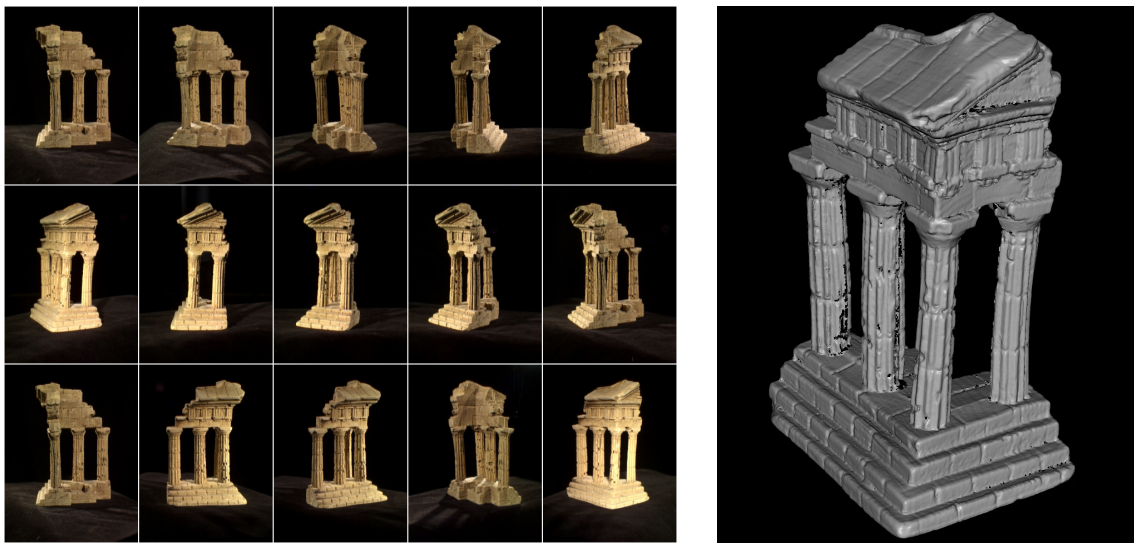
Other common vision problems that can be solved with the help of graph cuts are multiview reconstruction and shape fitting, see for instance [43, 44, 10].

Since the problem of calculating minimal cuts and maximum flows is a basic problem, a large number of algorithms for solving this problems have been created. The first algorithm dates back to Ford and Fulkerson in 1956 [24]. Their paper also contains the famous 'max-flow min-cut theorem', showing equivalence between these two problems. The algorithm of Ford and Fulkerson is only pseudo polynomial and the first polynomial algorithm was the algorithm of Edmunds and Karp [22]. During the next decades improved algorithms were presented until in 1986 Goldberg and Trajan presented their famous 'push-relabel algorithm' [28]. About ten years later Goldberg and Rao presented another algorithm [27] featuring the best running time until in 2013 Orlin presented an algorithm with superior runtime of $O(mn)$ [48].



(a) Segmentation example: in the image at the right the major regions are identified[1]

(b) Stereo correspondence: find corresponding pixels in the two slightly different input images. This allows the calculation of relative depth information[2]



(c) 3D reconstruction: calculate the 3D model from multiple images of the real world object[3]

Figure 0.0.: Three computer vision problems which can be modeled with graph cuts.

[1] image and ground truth taken from [46]
[2] image and ground truth taken from [55]
[3] image and ground truth taken from [57]

The above mentioned algorithms are designed in order to achieve good asymptotic runtimes for the general graph cut instances. Since in computer vision graph cuts often have grid structure, specialized algorithms for these problem instances were proposed. The algorithm of Boykov and Kolmogorov [8] is considered one of the fastest algorithms for such problems. Unfortunately, it is not known if the asymptotic runtime of this algorithm is polynomial [29].

Parallel to the development of these exact algorithms, approximation algorithms for max-flows min-cut were developed. A very basic approximation method was developed by Karger and Benczúr [34, 33, 5, 6]. Their approach is to reduce the number of edges in a clever way and then solve this modified problem exactly.

Recently, another class of approximation algorithms, using so called 'electrical flows', emerged [42, 19, 35, 45]. These algorithms depend on recent scientific results on approximately solving a special case of linear systems (to be precise: symmetric, diagonal dominant and positive definite linear systems) [61, 60, 59, 40, 38].

Another approach for solving graph cuts is to model them as continuous optimization problems and minimize this problem with classical methods from convex optimization. Compared to the previous methods, these algorithms are not specially designed for graph cuts, but they benefit from the fact that they can easily be parallelized and therefore are well suited to be used on GPUs [62].

For graph cuts with a special structure a solution can be obtained by solving the famous Rudin-Osher-Fatemi (ROF) functional [14]. In order to solve the ROF functional again common convex optimization algorithms can be used. Recently a new method called 'block coordinate descent' has been proposed. This method can easily be parallelized on CPUs and benefits from multicore CPUs [16].

In this thesis a short overview over two approximation algorithms utilizing electrical flows, namely the algorithm of Christiano et al. [19] and the algorithm of Lee et al. [42], is given. The algorithm of Christiano et al. was the first algorithm using electrical flows in order to compute max-flows and min-cuts. The algorithm of Lee et al. gives a nice interpretation of flows and cuts in conjunction with electrical flows. This is covered in the first part of the thesis. The second part is more about convex optimization and how graph cuts can be solved with algorithms of that particular field of optimization. All these algorithms are described and the building blocks, required by these algorithms, are provided. Besides the asymptotic runtime of these algorithms also their 'real' performance based on a C++ implementation is analysed. It turns out that, while the algorithms of Christiano et al. and Lee et al. have superior asymptotic runtime, the runtime for typical computer vision problem instances is not of any practical relevance.

The outline of the thesis is as follows: in chapter 1 the most basic mathematical definitions, used later in this thesis, are revised. Chapter 2 introduces electrical flows and some of the most important properties of electrical flows are described. Furthermore, it is shown how resistances influence electrical flows. Chapter 3 is dedicated to the algorithms of Christiano et al. (section 3.2) and the algorithms of Lee et al. (section 3.3). These algorithms are described and some experiments are performed and discussed to conclude the first part.

The second part starts with some basic definitions and facts about convex optimiza-

tion. Furthermore the equivalence between the binary min-cut problem and the continuous (relaxed) min-cut is shown (section 4.4). The primal-dual algorithm of Chambolle and Pock some of its modifications are presented in chapter 5. In chapter 6 it is discussed how the graph cut problem is a subproblem of the famous Rudin-Osher-Fatemi Total Variation problem. Furthermore, it is discussed how this ROF problem can be solved. The second part closes with a chapter comparing these algorithms to the very fast algorithms of Boykov and Kolmogorov [8].

Part I.

Electrical flow algorithms

1. Definitions and electrical flows

This chapter gives fundamental definitions of graph theory necessary to introduce the max-flow and the min-cut problem. Furthermore, it is explained how the asymptotic runtime of an algorithm is measured and how the quality of approximations is measured.

The experienced reader might skip this chapter, and fast-forward to chapter 2.

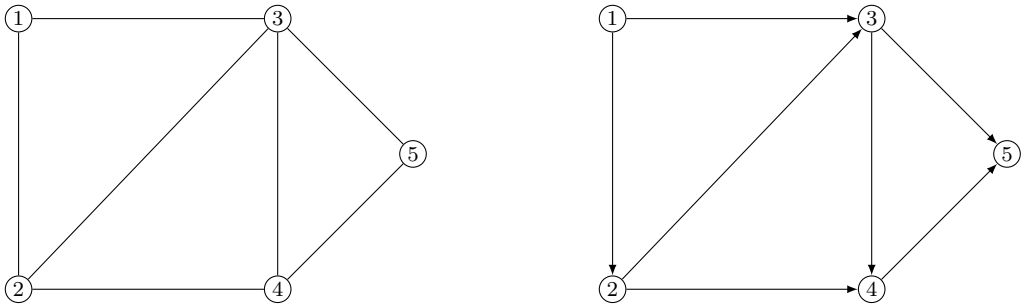
1.1. Basics

Starting with the very basics this section covers some definitions and notations of graphs and how they can be represented.

Definition 1. ((un-) directed graph)
Let V be a (finite) set, the so called *vertex set*. The set E containing unordered pairs of elements of V is called the *edge set*. The combination of these two sets is called a graph and is denoted as $G = (V, E)$.
In a directed graph the edge set E consists of ordered pairs, i.e. $E \subset V \times V$.

If G is a directed graph and $e = (u, v) \in E$, u resp. v is called the initial resp. terminal vertex of e. Unless specified otherwise by a 'graph' an undirected graph is meant.

In Figure 1.1 a visualization of a directed and undirected graph is shown.



(a) Undirected graph (b) Directed graph

Figure 1.1.: Visualization of a graph with vertex set $V = \{1, 2, 3, 4, 5\}$ and edge set $E = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\}$ in the directed case

If we want to refer explicitly to the edge set or vertex set of a graph G the following notation is used:

Notation 1. For a given graph G, $V(G)$ denotes the vertex set and $E(G)$ denotes the edge set of G.

Sometimes it is convenient to consider only a smaller part of a graph, not consisting of all vertices.

Definition 2. (Subgraph)
Let $G = (V, E)$, $S \subset V$ and $E' = E \cap (S \times S)$ then the (vertex) induced subgraph $G(S)$ is defined as $G(S) := (S, E')$

Definition 3. (neighbours, degree)
Given an undirected graph $G = (V, E)$, the neighbours of a vertex $v \in V$, denoted by $\delta(v)$, are all vertices sharing an edge with v i.e.

$$\delta(v) := \{w \in V | \exists e \in E : v \in e \wedge w \in e\}$$

The *degree* of a vertex v is the number of neighbours and is denoted by $\mathrm{d}(v)$, i.e. $\mathrm{d}(v) = |\delta(v)|$.
If G is a directed graph $\delta_+(v)$ is defined as all edges where v is the initial vertex and $\delta_-(v)$ as the set of edges where v is the terminal vertex i.e.

$$\delta_+(v) = \{(v, u) \in E | u \in V\} \qquad \delta_-(v) = \{(v, u) \in E | u \in V\}$$

therefore, the neighbours of v are both sets together: $\delta(v) = \delta_+(v) \cup \delta_-(v)$.

Two vertices v_i and v_j are called *adjacent* if there is an edge $\{v_i, v_j\} \in E$. Since the tuples of all adjacent vertices are exactly the edges of a graph one can define the so called *adjacency matrix* to represent a graph.

Definition 4. (Adjacency Matrix)
For a given graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$, the adjacency matrix $A \in \{0, 1\}^{n \times n}$ is defined as $A = (a_{ij})_{i,j=1}^{n}$ where

$$a_{ij} := \begin{cases} 1 & \text{if there is edge between } v_i \text{ and } v_j \\ 0 & \text{else} \end{cases}$$

The adjacency matrix is only defined for undirected graphs, therefore, the adjacency matrix of an directed graph is constructed by removing all the orientations.

A closely related property is the property of incidence. While adjacency is a property between two vertices, incidence is a property between an edge and a vertex. A vertex v is *incident* to an edge e if $v \in e$. Similar to the adjacency matrix one can define the s.c. *incidence matrix* by collecting all incidences of the graph:

Definition 5. (Incidence Matrix)
For a given graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$, the Incidence matrix $B \in \{0, -1, 1\}^{m \times n}$, $B = (b_{ij})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$ is defined as:

$$b_{ij} := \begin{cases} 1 & \text{if } e_i \in \delta_+(v_j) \\ -1 & \text{if } e_i \in \delta_-(v_j) \\ 0 & \text{else} \end{cases}$$

Note that in contrast to the adjacency matrix the incidence matrix is defined for directed graphs. A third matrix associated with a graph is the *Laplace matrix* which establishes a link between the adjacency matrix and the incidence matrix.

Definition 6. (Laplace matrix)
Let $G = (V, E)$ be a directed graph and B its incidence matrix then the Laplacian matrix L is defined as:

$$L := B^t B$$

An equivalent definition of the Laplacian, involving the adjacency matrix is given by:

$$L = D - A \tag{1.1}$$

where D is the s.c. degree matrix, which is the diagonal matrix of the vertex degrees, i.e. $D = \mathrm{diag}(\mathrm{d}(v_1), \cdots, \mathrm{d}(v_n))$.

There is a own theory studying the properties of these matrices. This theory is called *spectral graph theory* and links the properties of these matrices with properties of the corresponding graph.

Equation (1.1) indicates that the Laplacian matrix is diagonal dominant. Furthermore, L is symmetric since A is a symmetric matrix. The third property of the Laplacian is positive semidefiniteness. This can easily be observed by considering an eigenvalue λ and the corresponding (normalized) eigenvector v of L. Then

$$\lambda v = Lv \Leftrightarrow \lambda = v^t L v = v^t B^t B v = (Bv)^t Bv = \langle Bv, Bv \rangle \geq 0 \tag{1.2}$$

indicating that every eigenvalue is nonnegative which is the definition of positive semidefinite.
The properties symmetry, positive definiteness and diagonal dominance are abbreviated by the term sdd and such a matrix L is called a *sdd matrix*.

The number of nonzeros in each row of the adjacency matrix is exactly the degree of the vertex associated with this row. Since every nonzero entry is 1, one gets $\sum_{j=1}^{n} a_{ij} = \mathrm{d}(v_i)$ combining this with the observation from (1.1) it is clear that vector $(1, 1, \ldots, 1)^t \in \ker(L)$ and, therefore, L is singular.
One result of the previous mentioned spectral graph theory is that the graph G is connected iff the second smallest nonzero eigenvalue of its laplacian matrix is positive.

A singular matrix does not have a inverse matrix but there is the concept of the *pseudo inverse* matrix which has most of the properties of the inverse matrix but is defined for singular and nonquadratic matrices.

Definition 7. (Moore-Penrose pseudo inverse)
For a given Matrix $A \in \mathbb{R}^{n \times m}$ with $m, n \in \mathbb{N}$, the matrix A^+ with the following properties

- $AA^+A = A$
- $A^+AA^+ = A^+$

- $(A^+A)^t = A^+A$
- $(AA^+)^t = AA^+$

is called the Moore-Penrose pseudoinverse (or simply pseudoinverse) of A.

If $A \in \mathbb{R}^{n\times n}$ is a regular then $A^+ = A^{-1}$.

One important property of this pseudo inverse is that given a $b \in \mathrm{Im}(A)$, this x can be expressed as $b = Ay$ (for a suitable $y \in \mathbb{R}^m$). Now

$$\begin{aligned} AA^+Ay &= Ay \\ AA^+(Ay) &= (Ay) \\ AA^+b &= b \\ A(A^+b) &= b \end{aligned}$$

Thus one solution of the system $Ax = b$ is given by $x = A^+b$. Later the pseudoinverse of L will be used to solve linear systems of the Laplacian.

One algorithm presented later will heavily use the following fact of the pseudoinverse:

- $P = AA^+$ is a projection onto $\mathrm{Im}(A)$
- $\ker(A^+) = \ker(A^t)$
- $\mathrm{Im}(A^+) = \mathrm{Im}(A^t)$

In general we will denote norms by $\|\cdot\|$, with $\|\cdot\|_2$ being the euclidean norm, i.e. $\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$, where x is a element of some n dimensional vector space. The other two important norms, the infinity norm and the Manhattan norm are defined as usual: $\|x\|_\infty = \max_{i\in\{1,\dots,n\}} |x_i|$ and $\|x\|_1 = \sum_{i=1}^n |x_i|$.

In the following the set $X \subseteq \mathbb{R}^n$ will denote a simple subset, where n is of course finite.

Definition 8. (convex set)
The set X is called convex if

$$\lambda x + (1-\lambda)y \in X \quad \forall x, y \in X, \quad \forall \alpha \in [0,1]$$

Closely related is the definition of a convex function

Definition 9. (convex function)
Let $X \subseteq \mathbb{R}^n$ be a convex set, a function $f : X \to \mathbb{R}$ is called convex if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y) \qquad \forall x, y \in X, \quad \forall \alpha \in [0,1]$$

There are a lot of equivalent definitions and conditions for convex functions, we will not go into detail. For a comprehensive overview see for example [51].

The, at least for our cases, most important property of a convex function is the fact, that a local minima is equal to a global minima. In the general case, finding a global

optima of a function is a very challenging task and one cannot hope to find global solutions, but convexity makes this task actually traceable.

An additional property of a convex function is that, if the function is bounded on a convex set, it is continuous at the interior of this set. This is a kind of intuitive property since 'jumps' would violate the conditions in Definition 9 and for a closed interval $[a, b]$ the function $f(b) = 1$ and zero else is convex (on the interval) but certainly not continuous. A mathematical proof of this property can be found in [54].

Another important property of functions is Lipschitz continuity:

Definition 10. (Lipschitz continuity)
A function $f : X \to \mathbb{R}^n$ is called Lipschitz continuous if there exists a constant $L \in \mathbb{R}_+$, such that

$$\|f(x_1) - f(x_2)\| \leq L \|x_1 - x_2\|$$

The constant L is called the 'Lipschitz constant'.

As we will see later the speed of convergence of many optimization algorithms will depend on the Lipschitz constant of the objective function.

1.2. Flows and Cuts in a graph

It is often convenient to assign numerical values to the edges of a graph. This allows to prioritize or weight the edges, hence such a graph is called a weighted graph.

Definition 11. (weighted graph)
Let $G = (V, E)$ be a directed or undirected graph. A function $w : E \to \mathbb{R}$ is called *weight function* and the triple $G = (V, E, w)$ is called a weighted graph.

If there is an total ordering of the edges the weight function is often written in vector form $w := (w(e_1), w(e_2) \ldots, w(e_m))^t$. This vector w is called the *weight vector*.

Cuts

Now it is time to define a cut. For the sake of completeness we will start with the very basic definition of a cut.

Definition 12. (cut)
Let $G = (V, E)$ be a graph, a cut is a partition of the vertices into two nonempty sets, i.e. if $\emptyset \neq S \subset V$ then $(S, V \setminus S)$ is a cut of G. If it is clear from context we may simply say 'S is a cut of G'

An edge $(u, v) \in E$ is said to *cross* the cut iff u and v are in distinct sets. The *cross set* is the set of all edges crossing the cut.

Definition 13. (cut set/cross set)
The cut set or cross set of a graph $G = (V, E)$ is denoted by $C(S, V \setminus S)$ and defined as

$$C(S, V \setminus S) := \{(u, v) \in E | (u \in S \wedge v \notin S) \vee (u \notin S \wedge v \in S)\}$$

Again if it is clear from context one may write $C(S) := C(S, V \setminus S)$

Since the cross set of a cut is unique and fully defines a cut, depending on the context, the term 'cut' can refer to ether the 'vertices of a cut' or the cut set.

The previous definition of a cut is very basic and no specific properties of the underlying graph are used so far, therefore, if a weighted graph is considered the *value of a cut* is generally of interest:

Definition 14. (value of a cut)
Let $G = (V, E)$ be a graph and $S \subset V$ a cut. The value of a cut, denoted by $val(S, V \setminus S)$ or $val(S)$, is defined as the summed weights of the edges in the cross set.

$$val(S) = \sum_{(u,v) \in C(S)} w(u, v)$$

Another important definition in combination with cuts is the definition of a *s-t cut*. A s-t cut is a cut with two distinct vertices denoted by s and t where $s \in S$ and $t \notin S$. These s-t cuts are closely linked to the s.c. s-t flows, which will be defined later in this section (see Definition 17). Additionally these s-t flows have a nice physical interpretation compared to s-t cuts.

Definition 15. (s-t cut)
Given a graph $G = (V, E)$, and two distinct vertices $s, t \in V$. A st-cut $S \subset V$ is a cut where $s \in S \wedge t \notin S$.

At this point we have to stop for a moment and clarify some issues regarding directed graphs. Usually the value of a cut is defined as the weights of the directed edges starting at the set S and ending at the set $V \setminus S$. Furthermore, all weights are nonnegative.

In our case negative weights are not forbidden, therefore, all the edges between S and $V \setminus S$ have to be considered. The reason for this restrictions will become clear once flows, and especially flows in undirected graphs, have been defined.

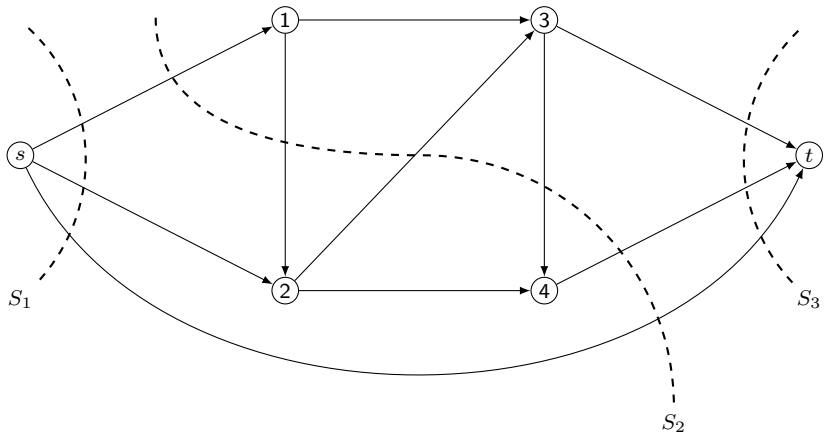


Figure 1.2.: Three different cuts: $S_1 = \{s\}$, $S_2 = \{s, 2, 4\}$, $S_3 = \{s, 1, 2, 3, 4\}$

Now all prerequisites for the central definition of this thesis are complete and it is time to define the *min-cut problem*.

Definition 16. (min-cut Problem)
Let $G = (V, E, w)$ be a weighted graph, $s, t \in V$, the min-cut problem is defined as

$$\min_{\substack{\emptyset \neq S \subset V \\ s \in S, t \notin S}} val(S) \tag{P_c}$$

In subsequent chapters further equivalent problem definitions will occur. This is due to the fact that the above definition is often not very convenient to handle, from an algorithmic point of view.

Flows

The concept of flows in a graph is probably easier to interpret than the concept of cuts. In the very simply case one can interpret the edges of the graph as water pipes and the vertices as connections of pipes. Therefore, the whole graph is a very simple water network. Now a certain amount of water is steadily inserted into the network at a source vertex s and removed at a sink vertex t. The water now flows from s to t and a certain fraction (possibly nothing) of the inserted water flows through each pipe. By the term 'flow' the distribution of the water over all the pipes is meant. On the other hand by congesting or removing certain pipes, one could prevent any water flowing from s to t. These pipes would define one possible cut set.

After this informal description a bit more formalism has to be introduced. In our example it was implicitly assumed that no water is lost inside the network and no water is accumulated at the connections. This property is called *flow preservation* and is essential for the basic definition of a flow:

Definition 17. (s-t Flow, flow preservation)
Given a directed graph $G = (V, E)$, and designated vertices $s, t \in V$ a s-t flow between

s and t is a function $f : E \to \mathbb{R}$, fulfilling the s.c. flow preservation defined as:

$$\forall v \in V \setminus \{s,t\} : \sum_{e \in \delta_-(v)} f(e) = \sum_{e \in \delta_+(v)} f(e)$$
$$\sum_{e \in \delta_-(s)} f(e) = \sum_{e \in \delta_+(t)} f(e)$$

The *value* of a st-flow is given by the sum of the flow outgoing from the source and denoted by F, i.e.

$$val(f) = \sum_{e \in \delta_-(s)} f(e)$$

Sometimes in the literature a flow is not allowed to have negative values but we explicitly allow for negative values. Furthermore, the definition of the flow preservation property and the value of a flow require a directed graph.
In this thesis undirected graphs are considered in most chapters and, therefore, this definition has to be extended or a way of converting problems in undirected graphs into problems in directed graphs has to be found.

One possibility to model flows in undirected graphs would be to simply duplicate each edge and assign them different orientations. But now it would be possible to send an arbitrary amount of flow through an edge (u, v) and return the flow immediately through the associated edge (v, u). This would render the previous definition useless, since the value of this flow could be infinite.

Similar to the standard flows for directed graphs only flows with nonnegative values are considered in undirected graphs. Therefore, each edge is assigned an arbitrary orientation and if an edge e has a negative flow this edge is considered orientated in the opposite direction. The nonnegative flow of an undirected graph $G = (V, E)$ is obtained by taking the absolute value (of the flow in the associated arbitrarily orientated graph).

Similar to the weight vector (and by a slight abuse of notation), the flow vector for a given graph $G = (V, E)$ and a flow f is defined as $f := (f(e_1), f(e_2), \cdots, f(e_m))$.

Now that f is a vector, the flow preservation property can be written as linear system. Given a graph $G = (V, E)$ and a vectors f, then f is a st-flow with value F iff

$$B^t f = F(\chi_s - \chi_t) \tag{1.3}$$

where B is the incidence matrix, χ_k is the k-th canonical unit vector, i.e. χ_k is 1 in the k-th component and 0 elsewhere.

There will be situations, where flow is not only inserted/removed at two distinct vertices but at possible every vertex. In this case the vector c_{ext} is used to describe the amount of flow inserted or removed. If $(c_{ext})_i$ is positive resp. negative then flow is inserted resp. removed at vertex i. The amount of flow entering the graph at a specific vertex v is called the 'inflow/outflow at vertex v'.

In the example with the water flowing through some pipes there was no restriction on the capacities of the pipes. The total amount of water may could flow through a single

pipe. It is easy to imagine that this may not be very realistic since real world pipes have a maximum capacity, restricting the flow through this pipe (e.g. diameter).

In order to model these observation in the graph, *capacities* are assigned to each edge.

Definition 18. (capacitated flow)
Let $G = (V, E)$ and $c : E \to \mathbb{R}$ s.c. capacity function, then f is a capacitated flow if it is a flow as in Definition 17 and, furthermore,

- $f : E \to \mathbb{R}_{\geq 0}$
- $\forall e \in E : \quad f(e) \leq w(e)$

An 'uncapacitated' flow is a capacitated flow where each weight is 1.

This type of flow is often called a *combinatorial flow* in order to distinguish it from a electrical flow, introduced in section 2.

Similar to the flow vector the capacity vector is defined as $c := (c(e_1), \ldots, c(e_m))$. Unless not specified otherwise, we will always consider capacitated flows in connection with weighted graphs.

Getting back to the initial example it is now possible to ask the question: What is the maximum amount of water able to flow through the pipe network? This leads to the famous *max-flow Problem*:

Definition 19. (Max-Flow Problem)
Given a weighted graph $G = (V, E, w)$, the Max-Flow Problem is defined as:

$$\max_{f \text{ st-flow}} val(f) \tag{P_f}$$

If f^* is a flow maximizing the flow value then f^* is called a maximal flow.

The problem of finding a maximal flow is closely related to the problem of finding a minimal cut. This link is established be the famous max-flow-min-cut theorem.

Theorem 1 (max-flow-min-cut-theorem [24, 23])**.** *Given a directed and weighted graph $G = (V, E, w)$ and two distinct vertices s and t, then the value of the maximal s-t flow is equal to the value of the minimal s-t cut.*

Note that the max-flow-min-cut theorem is formulated for *directed* graphs!

1.3. Landau-Notation, Approximation

Since we will deal with approximation algorithm one has to clarify what is meant by an 'approximation' and how one can compare different algorithms. Both terms will be described using the Landau notation, defined as follows

Definition 20. (Landau notation)
Let $f, g : \mathbb{R} \to \mathbb{R}$ be two functions, then:

(i) $f(n) \in O(g(n)) \iff \limsup_{n\to\infty} \left|\frac{f(n)}{g(n}\right| < \infty$

(ii) $f(n) \in \Omega(g(n)) \iff \liminf_{n\to\infty} \left|\frac{f(n)}{g(n}\right| > 0$

(iii) $f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$

(iv) $f(n) \in o(g(n)) \iff \limsup_{n\to\infty} \left|\frac{f(n)}{g(n}\right| = 0$

In literature it is sometimes common to write $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$. The notation in (i) is commonly referred as *big O notation*.

The most basic elements an algorithm is working with are simple numbers. An algorithm performs simple operations on these numbers, i.e. typical operations are comparisons or arithmetic operations. It is assumed that each of these operations takes constant time, regardless of the value of the involved number. In order to analyze the performance of an algorithm the number of operations is determined and this value is called the exact runtime of an algorithm. The number of operations will depend of course on the size of the problem instance, therefore, the runtime is a function of the problem size. To simplify the analysis the Landau-Notation is used and for an algorithm with runtime $f(n) \in O(g(n))$ it is said that the *asymptotic runtime* of this algorithm is $O(g(n))$. In general by *runtime* of an algorithm the asymptotic runtime is meant.

The algorithms presented in this thesis will often have a runtime of the form $O(n \log^c(n))$ for some constant c. Since the $\log(n)^c$ factor is of minor interest (compared to the n factor) the following notation is used to hide it.

Definition 21. ($\tilde{O}$-Notation)

$$f(n) \in \tilde{O}(g(n)) \iff f(n) \in O(g(n) \log^c(n)) \text{ for some constant } c$$

Having discussed running times of algorithms we can proceed by discussing the notation of approximation.

Given a reel valued function $f(x)$, called the objective function, which should be minimized (or maximized), i.e. find a $\bar{x}$ such that $f(\bar{x}) \leq f(x)$ for all x. An approximation (or approximate solution) of f is a $\tilde{x}$ such that $f(\tilde{x})$ is close to $f(\bar{x})$. The error is the difference between the value of this approximation and the optimal solution. Sometimes a distinction between absolute and relative error is made. The absolute error is simply given by $|f(\tilde{x}) - f(\bar{x})|$, whereas the relative error is given by $|{}^{(f(\tilde{x})-f(\bar{x}))}/_{f(\bar{x})}|$.

Definition 22. ($(1 \pm \varepsilon)$-approximation)
For a real valued function f with optimal solution $\bar{x}$ and a $\varepsilon > 0$. A vector $\tilde{x}$ is called a

- $(1+\varepsilon)$-approximation iff $\bar{x}$ is a minima and
$$f(\tilde{x}) \leq (1 + O(\varepsilon)) f(\bar{x})$$
- $(1-\varepsilon)$-approximation iff $\bar{x}$ is a maxima and
$$f(\tilde{x}) \geq (1 - O(\varepsilon)) f(\bar{x})$$

Note that a $(1\pm\varepsilon)$-approximation is only asymptotically close to the optimal solution. If approximation algorithms are compared their running time will depend on the problem size and on ε, i.e. the runtime looks like $O(f(n)g(\varepsilon))$. This justifies the O-Notation in the definition of the $(1+\varepsilon)$-approximation, since $O(\varepsilon)$ contains all functions of the form $c\varepsilon$ for constants $c > 0$ and by computing a solution with parameter ε/c the approximate solution will be not worse than $(1 \pm \varepsilon)f(\bar{x})$. Of course this hold only as long as $g(\cdot)$ is not too bad (think of $g(x) = x^x$), but in this thesis $g(x)$ is always x^c which does not cause any problems.

Nevertheless we will often write $(1+O(\varepsilon))$-approximation to emphasize the asymptotic dependence on ε.

2. Electrical Flows

The algorithms presented in the subsequent chapters have one thing in common. They heavily depend on some new results about solving linear systems of symmetric, diagonal dominant matrices. As previously observed, the Laplacian matrix of a graph has these properties.
In the following section a certain type of flows, the s.c. *electrical flows* are presented. These flows correspond to solutions of Laplacian systems, meaning they are a solution of a linear system involving the Laplacian matrix.
Unfortunately these flows do not fulfill any capacity constraints defined in the previous section, still they fulfill the flow preservations (otherwise they would not be called flows).

The name electrical flow originates from the fact, that these flows model the distribution of the currents in an electrical network[1]. Therefore, the edges are not seen as water pipes any more, but are now treated as electrical conductors and each conductor has a certain electrical resistance limiting the amount of electrical current, which is able to flow through the conductor. In the field of physics there are two laws describing these basic electrical flows. The first law is Kirchhoff's law and the second one is Ohm's law. In the following some necessary definitions in order to formulate these two laws are introduced.

2.1. Preliminaries, Definitions and Limitations

Since a electrical current can flow in both directions of a conductor[2], we can restrict ourself to *undirected* graphs. This is the reason why all flows were defined in undirected graphs in chapter 1.

As mentioned in chapter 1 all undirected graphs are implicitly converted into directed graphs by assigning a arbitrarily orientation to the edges.

Since electrical flows emerged from the field of physics, we will stick with their terms and notations and we will define the following functions:

Definition 23. (resistance, current, voltage, potential)

$r : E \to \mathbb{R}_+$	the resistance function
$f : E \to \mathbb{R}$	the current function
$\phi : V \to \mathbb{R}$	the voltage function

[1] at least in a very simple model
[2] at least in this simple model

the voltage function is also called the potential function.

Notation 2. Similar to the definition of the flow vector, the resistance vector is defined as $r := (r(e_1), r(e_2), \cdots, r(e_m))$ and analog for the current vector i and the voltage vector u.

Notation 3. Very often potential differences (across a edge) will be used, i.e. $f(v)-f(w)$ for a edge $(v, w) \in E$. Instead of defining a new function the function ϕ is overloaded for edges $e = (u, v)$ by: $\phi(e) = \phi(u) - \phi(v)$. This overloaded function ϕ is called the vector of potential differences.

As previously mentioned, voltage is inserted/removed at some vertices. This is covered by the vector $c_{ext} \in \mathbb{R}^n$ where the value at i-th position of c_{ext} denotes the amount of voltage inserted (if the value is positive) or removed (if value is negative) at vertex v_i. If a current of value F is inserted only at vertex s and removed at vertex t then, similar as in the previous section, χ instead of c_{ext} is written.

Modeling Kirchhoff's and Ohm's law

There are actually two laws named after Kirchhoff. The first one is called *Kirchhoff's current law* and the second one *Kirchhoff's voltage law* or sometimes simply Kirchhoff's first and second law. Kirchhoff's current law simply says that the total amount of current flowing into a vertex is equal to the total amount of current flowing out of a vertex. This is exactly the flow preservations from Definition 17 and sometimes also called the principle of 'conservation of electric charge'. Kirchhoff's potential law or the 'principle of conservation of energy' states that the sum of potential differences around a circle is zero.

Kirchhoff's laws describe the current and voltage inside a graph but they do not link these two entities together. This link is established by Ohm's law. Ohm's law is described by the famous equation: $I = {}^{V}/_{R}$. Where I resp. R is the current resp. resistance of an conductor and V is the potential difference across this conductor, i.e. the voltage difference of the endpoints of the edge.

Since Kirchhoff's current law corresponds to the flow preservation, one can simply reuse the property from Definition (1.3), which in terms of currents and resistances reads:

$$B^t f = c_{ext} \tag{2.1}$$

As mentioned above, Ohm's law states that the flow on each edge is the potential difference divided by the resistance. In other words the following equation has to hold:

$$\forall e = (v, w) \in E : \quad f(e) = \frac{\phi(v) - \phi(w)}{r(e)}$$

or written as Matrix-Vector product:

$$f = \operatorname{diag}(r)^{-1} B\phi \tag{2.2}$$

Notation 4. Given a graph $G = (V, E)$ and a resistance vector r then the tuple (G, r) is called a electrical network.

Definition 24. (electrical flow)
Given a electrical network (G, R) and a vector c_{ext}. Then the the flow f is a electrical flow if equations (2.1) and (2.2) for a ϕ are fulfilled.

For the sake of readability let $R := \text{diag}(r)$ and by combining both equations (2.1) and (2.2) one gets:

$$B^t R^{-1} B\phi = c_{ext} \tag{2.3}$$

Now if all resistances would be equal to 1 the matrix one the left-hand side of (2.3) is exactly the Laplacian of the graph. Since R is a diagonal matrix the matrix $B^t R^{-1} B$ could be interpreted as a reweighted version of the Laplacian matrix L. This motivates the definition of the *weighted Laplacian*:

Definition 25. (weighted Laplacian)
Let $G = (V, E)$ be a graph with *weights* $w \in \mathbb{R}^{|E|}$, B the incidence matrix and $W = \text{diag}(w)$ the diagonal matrix of the weights. Then the weighted Laplacian matrix or weighted Laplacian of G is defined as:

$$L = B^t W B$$

Note that both, the Laplacian and the weighted Laplacian are denoted by L, but this will not be a problem since we will treat the Laplacian of unweighted graphs as weighted Laplacians with weights 1.

It is straightforward to check that the Laplacian matrix is given by:

$$L_{ij} = \begin{cases} \sum_{e \in \delta_-(i) \cup \delta_+(i)} w_e & \text{if } i = j \\ -w_e & \text{if } (i,j) \in E \vee (j,i) \in E \\ 0 & \text{else} \end{cases}$$

It is easily observed that the weighted Laplacian is again symmetrically and diagonal dominant. If the weights are non-negative and the same argument used as in (1.2) it can be shown that the weighted Laplacian is also positive semidefinite.

Now one can write (2.3) in term of the Laplacian Matrix and it becomes clear that computing an electrical flow is equivalent to solve the following linear system:

$$\begin{aligned} & L\phi = c_{ext} \\ \Leftrightarrow \quad & \phi = L^+ c_{ext} \end{aligned} \tag{2.4}$$

Recently some groundbreaking results about approximately solving such sdd systems were made by Spielman and Teng [60] and later simplified and improved by Koutis, Miller and Peng [38, 39].

Theorem 2 (Koutis et al. [38]). *Let A be a $(n \times n)$ sdd matrix with m nonzero entries, $b \in \mathbb{R}^n$ a vector and $x \in R^n$ a unknown vector fulfilling $Ax = b$. Then there exists an algorithm calculating a vector $\tilde{x}$ such that $\|x - \tilde{x}\|_A < \varepsilon \|\tilde{x}\|_A$ for some $\varepsilon > 0$. Furthermore, this algorithm has runtime $\tilde{O}(m \log(n) \log(1/\varepsilon))$.*

The term $\|\cdot\|_A$ denotes the vector norm induced by the matrix A, i.e. $\|x\|_A = \langle x, x \rangle_A = x^T Ax$.

Energy in electrical flows

The *energy* of an edge $e = (v, w) \in E$ is defined as ${}^{(\phi(v)-\phi(w))^2}/_{r(e)} = f(e)^2 r(e) = (\phi(v) - \phi(w)) f(e))$. The total energy of a electrical network is simply the sum over all these energies:

Definition 26. (total energy)
Given a electrical network (G, r). Let f be a flow fulfilling the flow preservation constraints. The total energy of f, $\mathcal{E}_r^G(f)$ is defined as the sum over all the energies, i.e.

$$\mathcal{E}_r^G(f) := \sum_e r_e f_e^2 = f^t R f = \left\| R^{1/2} f \right\|_2^2$$

To simplify the notations, the superscript will be dropped if it is clear which graph is meant. Furthermore, if it is clear from context 'energy of f' instead of 'total energy of f' will be written.

If f is a electrical flow then the energy of an edge can be written in terms of potential differences as well as in terms of current. This allows to express the total energy as:

$$\mathcal{E}_r(f) = \sum {}^{(\phi_u - \phi_v)^2}/_{r_e} = \sum_{e=(u,v)} f(u, v)(\phi(u) - \phi(v))$$

The next theorem is a important one since it states that the electrical flow minimizes the energy in an electrical network.

Theorem 3 (Dirichlet Principle, optimality of electrical flows). *Let (G, r) be a electrical network, c_{ext} the vector of sources/sinks. Then there exists a flow f, such that*

- $f_i = (c_{ext})_i \qquad \forall i$ *with* $(c_{ext})_i \neq 0$
- f *satisfies Ohm's law*
- $f = \arg\min_{\bar{f} flow, val(\bar{f})=F} \mathcal{E}(\bar{f})$

For a proof see [7].

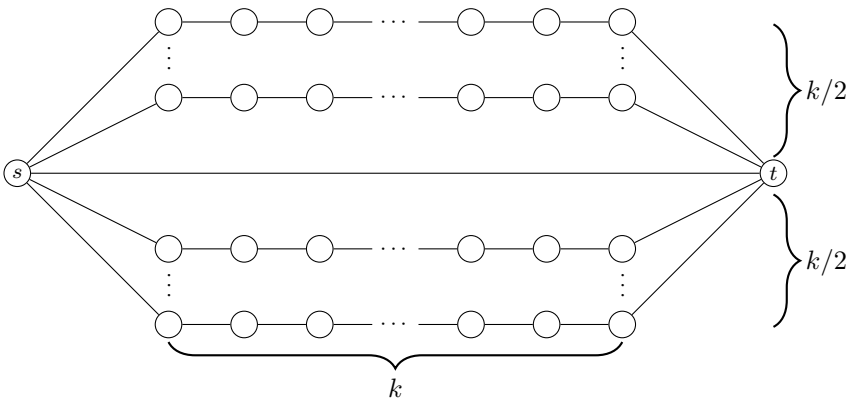


Figure 2.1.: If all resistances and capacities are equal to one, the combinatorial flow on every edges is 1. In the electrical flow the flow at the edge in the middle is k. Therefore, the electrical flow violates the capacity constraint by a factor of $O(\sqrt{m})$.

2.2. From electrical flow to combinatorial flows

In this section we discuss how electrical flows can be used to compute combinatorial flows.

As observed previously, both types of flow share the flow preservation constraints but Ohm's law and capacity constraints are different concepts.

The probably most intuitive approach is to set the resistance of an edge e to $1/w_e$. This is justified by the rule 'more resistance less flow' but it is not enough to respect the capacity constraints. This is not surprising since electrical flows minimize the energy, which is a quadratic function, while (P_f) is linear. A worst case example is given in Figure 2.1. The maximal combinatorial flow is k and on each edge the flow is one. The electrical flow with value k will send $k/2$ current over the edge in the middle. Since $k = \sqrt{m-2}$ on a single edge the difference between the electrical and the combinatorial flow can be $\Theta(\sqrt{m})$.

In the following section a few results, on how changing the resistances will influence the electrical flow, are presented.

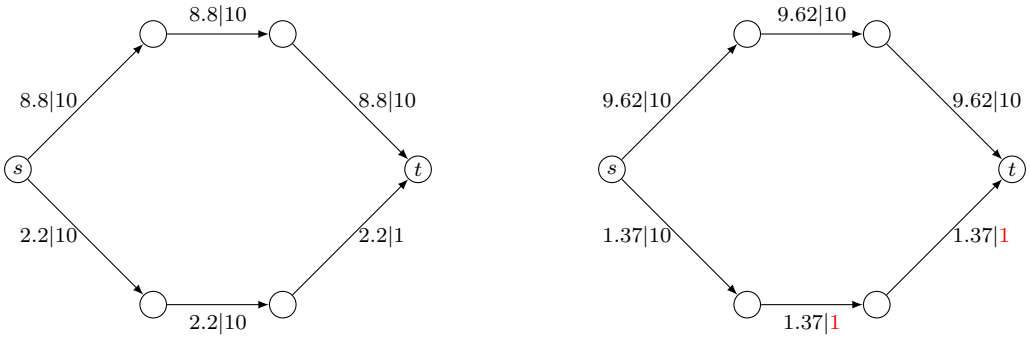
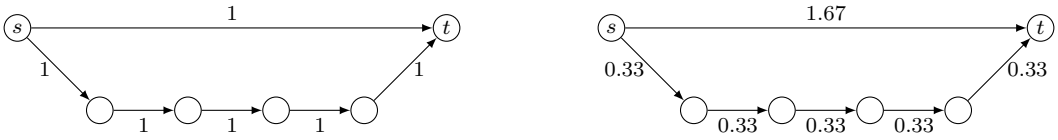


Figure 2.2.: Difference between electrical flows of different resistances



(a) Combinatorial flow of value 2

(b) Electrical flow value 2

Figure 2.3.: Difference between electrical flow and combinatorial flow

2.2.1. Controlling resistances

The first theorem states that one can replace a electrical network with source s and sink t by a single edge (s,t) without changing the energy in the network.

Theorem 4 (Rayleigh's principle)**.** *Let f be a s-t flow with value F. Let $g : V \to \mathbb{R}$ be a arbitrary function on the vertices, then*

$$(g(s) - g(t))F = \sum_{(u,v)\in E} (g(u) - g(v))f(u,v)$$

Proof. Writing the right-hand side in terms of vertices, one gets:

$$\sum_{x\in V} g(x) \underbrace{\left(\sum_{y\in\delta_+(x)} f(x,y) - \sum_{y\in\delta_-(x)} f(y,x) \right)}_{0 \text{ if } x\neq s,t} = g(s)F - g(t)F$$

□

In the context of electrical flows this result states that one can replace the whole s-t network by a single edge without changing the potential. Hence, this theorem is sometimes called the 'conservation of energy principle'.

Definition 27. (effective resistance/effective conductance)
Let (G, R) be a elecrical network and ϕ a potential vector of some unit electrical flow f, i.e. the potential difference between ϕ_s and ϕ_t is one, then the effective conductance $\mathrm{C}_{\mathrm{eff}}$ is defined as the amount of current flowing form s to t. The effective resistance denoted by $\mathrm{R}_{\mathrm{eff}}$ is $\mathrm{R}_{\mathrm{eff}} = 1/\mathrm{C}_{\mathrm{eff}}$.

Writing the total energy of a electrical flow in terms of its potentials then it follows from theorem 4 that the total energy is equal to $(\phi_s - \phi_t) \sum_{v \in \delta(s)} f(s, v)$.

Now if the potential difference between s and t is set to one, i.e. $\phi_s - \phi_t = 1$, then it follows that the total energy, the flow value and the effective conductance have the same value.
On the other hand if the flow value is 1 then it follows that the total energy, the potential difference and the effective resistance have the same value. This can be seen by the following argument: consider the flow f with value 1, then

$$\mathcal{E}(f) := f^t R f = \phi^t B^t R^{-1} B \phi = \phi^t L \phi = \phi^t \chi = \phi(s) - \phi(t)$$

The first equation is due to equation (2.2) and the third due to (2.4).

Now it immediately follows:

Corollary 1. *Let $G = (V, E)$ and $r \in \mathbb{R}^m$ be a resistance vector, then*

$$\mathrm{C}_{\mathit{eff}}(r) = \min_{\phi | \phi_s = 1, \phi_t = 0} \sum_{e=(u,v) \in E} \frac{(\phi_u - \phi_v)^2}{r_e}$$

$$\mathrm{R}_{\mathit{eff}}(r) = \min_{f \text{ flow}, val f = 1} \sum_{e=(u,v) \in E} f(e)^2 r_e$$

The previous corollary basically says that: If the resistance of an edge is increased, the effective resistance does not decrease. If the two vertices are shorted , the effective resistance does not increase.

The next lemma answers the question how much the effective resistance changes if one modifies the resistance of an edge.

Lemma 1. *Let f be an electrical flow, r its resistance vector and $h = (u, v) \in E$ with*

$$f_h^2 r_h = \beta \, \mathrm{R}_{\mathit{eff}}(r)$$

If the resistance of edge h is modified by a multiple of γ,

$$r' := \begin{cases} \gamma r_e & \text{if } h = e \\ r_e & \text{else} \end{cases}$$

then

$$\mathrm{R}_{\mathit{eff}}(r') \geq \frac{\gamma}{\beta + \gamma(1 - \beta)} \mathrm{R}_{\mathit{eff}}(r)$$

In case of cutting this edge which is equivalent to $\gamma = \infty$ the change is $\mathrm{R}_{\mathit{eff}}(r') \geq \frac{Reff(r)}{1-\beta}$.
In case of $\gamma = 1 + \varepsilon$ with $\varepsilon \leq 1$ the change is $\mathrm{R}_{\mathit{eff}}(r') \geq \frac{1+\varepsilon}{\beta + (1+\varepsilon)(1-\beta)} \mathrm{R}_{\mathit{eff}} \geq (1 + \frac{\beta\varepsilon}{2}) \mathrm{R}_{\mathit{eff}}(r)$

The proof is quite technical and does not offer any specific insights, therefore, we refer to [19].

3. Algorithms

In this chapter the algorithm of Christiano et al. and the algorithm of Lee et al. are presented. The algorithm of Christiano et al. was the first algorithm using electrical flows in order to approximately solve max-flow and min-cut problems. The algorithm of Lee et al. uses a totally different approach and has some very nice interpretations in terms of projections to cycle and vertex space of the underlying graphs. For both algorithms the main results are presented and afterwards some experiments are conducted. Unfortunately, the runtime of both algorithm has very large constants (which are hidden by the O-Notation) and these constants make the algorithms inefficient for typical computer vision problems, as it will be shown by the experiments.

3.1. Overview and preliminaries

The main outline of both algorithms is practically the same. In this section the steps for the min-cut algorithms are discussed (the steps for the max-flow algorithms are quite similar).

First the algorithms reduce the number of edges in the graph such that the graph gets sparse. In the next step the remaining edge weights are modified such that the ratio between largest and smallest weight is (polynomially) bounded. Furthermore an upper and a lower bound on the maximum flow value is calculated which gives an interval containing the optimal value. This interval is searched with binary search, in order to get a approximate solution. In order to perform this binary search, the algorithms use a subroutine which answers the question whether a certain amount of flow can be routed through the graph and if true returns a valid flow . This subroutine is the only difference between the two algorithms. The basic outline of both algorithms looks like:

1. Random sample the graph
2. scale capacities
3. calculate a upper bound $\bar{F}$ lower bound $\underline{F}$
4. perform binary search on $[\underline{F}, \bar{F}]$

The 'random sampling' step reduces the number of edges from at most $O(n^2)$ to $O(n \log(n))$, while the cut values in this new graph do not change too much. This is achieved by constructing a new sparse graph with the same vertex set but a different edge set. While this technique is useful for the calculation of approximate min-cuts, it is not known how to convert a valid flow in the spare graph to a valid flow in the original graph.

Therefore, this step is only feasible for min-cut algorithms. Bounding the ratio between the smallest and the largest weight ensures that the difference between lower and upper bound is not too large. Furthermore, this reduces the number of binary search steps to $O(\log(n))$ steps. Since, this number occurs as a multiplicative factor in the total runtime it does not affect the runtime measured in terms of the $\tilde{O}$-Notation. All these common steps will be discussed in detail in the rest of this section. Additionally a method for quickly approximating electrical flows will be presented, since both algorithms heavily depend on this property.

3.1.1. Computing electrical flows

In this section it is outlined how to convert an approximate electrical flow into a valid electrical flow. This is an important routine for the approximation algorithms relying on electrical flow computations. As stated by Theorem 2 one can very efficiently calculate an approximation of the electrical flow. In general this approximation is not a valid electrical flow nor a valid flow.

In the following section it is assumed that (G, r) is a electrical network (i.e. $G = (V, E, w)$ with $w = 1/r$ is a weighted graph) and χ_{st} the vector of the induced potentials at s and t. As in chapter 1 the incidence matrix is denoted by B and the weighted Laplacian by L. Furthermore, let f be the exact electrical flow of this network and F its flow value. The resistances are assumed to lie in the interval $[1, R]$ (otherwise they can be scaled).

For their algorithm, Christiano et al. described a method to obtain a valid electrical flow from an approximation calculated, by the approximation algorithm of Koutis et al., stated in Theorem 2.

Theorem 5 (Fast Approximation of Electrical Flows (Christiano et al. [19]))**.** *Let $\delta > 0$, $F > 0$ and $r \in \mathbb{R}^m$ be the vector of resistances with ratio between largest and smallest resistance $R := \max(r_i)/\min(r_i)$. The electrical flow with flow value F is denoted by f. In time $\tilde{O}(m \log R/\delta)$, on can compute a vector of vertex potentials $\tilde{\phi}$ and the corresponding s-t flow $\tilde{f}$ with value F fulfilling the following properties:*

$$(i) \qquad \mathcal{E}(\tilde{f}) \leq (1+\delta)\,\mathcal{E}(f)$$

$$(ii) \qquad \forall e \in E \qquad \left| r_e f_e^2 - r_e \tilde{f}_e^2 \right| \leq \frac{\delta}{2mR}\,\mathcal{E}(f)$$

$$(iii) \qquad \tilde{\phi}_s - \tilde{\phi}_t \geq \left(1 - \frac{\delta}{12nmR}\right) F\,\mathrm{R}_{\mathit{eff}}(r)$$

In the following the proof is given for property (*i*). The reason is that this proof is constructive and the method to obtain such a flow is described. The proofs for (*ii*) and (*iii*) are more technical and are omitted.

Proof. Since all the resistances lie between 1 and R and the value of the flow f is F one has the following inequalities

$$\frac{F^2}{m} \leq \mathcal{E}(f) \leq F^2 R m$$

Recalling the definition of $\mathcal{E}$ (given by Definition 26), the first inequality follows by lower bounding the resistances by 1 and applying Hölder's inequality. The second inequality follows from trivially bounding the sum.

Recall from chapter 2 that the potentials ϕ of the electrical flow are given by $L\phi = F\chi$. Now the result of Koutis et al. is used to compute a vector $\hat{\phi}$ such that

$$\left\|\hat{\phi} - \phi\right\|_L \leq \varepsilon \left\|\phi\right\|_L$$

The approximated flow $\hat{f}$ is given by $\hat{f} = R^{-1}B^t\hat{\phi}$, but in general $B\hat{f} \neq F\chi_{st}$.

A convenient fact is that the energy and the potentials are linked by $\left\|\phi\right\|_L^2 := \phi^t L\phi = \mathcal{E}(f)$. This allows to easily bound the energy of $\hat{f}$:

$$\mathcal{E}(\hat{f}) = \left\|\hat{\phi}\right\|_L^2 \leq \left(\left\|\phi\right\|_L + \left\|\hat{\phi} - \phi\right\|_L\right)^2 \leq (1+\varepsilon)^2 \left\|\phi\right\|_L^2 = (1+\varepsilon)^2 \mathcal{E}(f)$$

For the flow $\hat{f}$ the overflow at each vertex is given by $c_{ext} := B\hat{f}$. Since $\langle c_{ext}, 1\rangle = 0$, c_{ext} can be seen as a demand vector to a flow problem with multiple sources/sinks, therefore, the task is to route the overflow given by c_{ext}. The difference of $\hat{f}$ and this routing flow is exactly a valid flow with flow value F. This new flow is denoted by $\tilde{f}$ and obeys the flow preservation constraints:

$$B^t\tilde{f} = F\chi_{st}$$

The maximal amount of flow one has to reroute from/to a vertex is given by

$$\eta := \left\|c_{ext} - F\chi_{st}\right\|_\infty \leq \left\|c_{ext} - F\chi_{st}\right\|_2 = \left\|L\hat{\phi} - L\phi\right\|_2 \leq \left\|L\right\|_2 \left\|\hat{\phi} - \phi\right\|_L \leq 2n\varepsilon\sqrt{\mathcal{E}(f)}$$

It is clear that this flow is bounded by $n\eta$ and in order to compute such a flow it is sufficient to consider the flow in a spanning tree of G. In a spanning tree T such a flow can be computed in linear time.

Now the only thing left to show is that the energy of $\tilde{f}$ really is a $(1+\delta)$-approximation

of f.

$$\begin{aligned}
\mathcal{E}(\tilde{f}) &= \sum_e r_e \tilde{f}_e^2 \\
&\leq \sum r_e (\tilde{f}_e + n\eta)^2 \\
&\leq \mathcal{E}(\hat{f}) + (2n\eta F + n^2\eta^2) \sum_e r_e \\
&\leq \mathcal{E}(\hat{f}) + \left[2nF(2n\varepsilon\sqrt{\mathcal{E}(f)}) + n^2 4n^2\varepsilon^2\, \mathcal{E}(f)\right] mR \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) \left[4n^2 \frac{1}{\sqrt{\mathcal{E}(f)}} + 4n^4\varepsilon\right] mR \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) \left[4n^2 F \frac{\sqrt{m}}{F} + 4n^4\varepsilon\right] mR \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) \left[4n^2\sqrt{m} + 4n^4\varepsilon\right] mR \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) \left[4n^3 + 4n^4\varepsilon\right] mR \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) 4n^4 mR \left[\frac{1}{n} + \varepsilon\right] \\
&\leq \mathcal{E}(\hat{f}) + \varepsilon\, \mathcal{E}(f) 4n^4 mR \\
&\leq \mathcal{E}(f)((1+\varepsilon)^2 + \varepsilon 6n^4 mR^{3/2})
\end{aligned}$$

In order to ensure that this new flow $\tilde{f}$ is a $(1+\delta)$-approximation ε has to be chosen as

$$\varepsilon = \frac{\delta}{12n^4 mR^{3/2}} \tag{3.1}$$

which yields:

$$\begin{aligned}
\mathcal{E}(\tilde{f}) &= \mathcal{E}(f) \left(1 + 2\frac{\delta}{12n^4 mR^{3/2}} + (\frac{\delta}{12n^4 mR^{3/2}})^2 + \delta/2\right) \\
&\leq \mathcal{E}(f)(1+\delta)
\end{aligned}$$

For the desired runtime simply plug the value of ε, given in (3.1), into Theorem 2. □

3.1.2. Random Sampling

This section gives a short overview of the 'random sampling' methods of Benczúr and Karger [6]. The first part is the actual random sampling method. This method allows the reduction of the edges to $O(n \log(n))$ for an arbitrary graph while nearly preserving the cut values. The second part is a sampling algorithm for computing maximum flows on graphs with small total flow.

Graph compression

When connected graphs are considered the number of edges m can be quadratic in the number of vertices n, i.e. $m \in O(n^2)$. The random sampling method of Benczúr and Karger allows to construct a new graph $\tilde{G}$ with n vertices and $O(n \log(n))$ edges with similar cut properties. If S is a cut with value F in the original graph, then the value of the cut in $\tilde{G}$ is at most $(1+\varepsilon)F$. Since this holds for all cuts, every exact min-cut algorithm can be converted into a $(1+\varepsilon)$-approximation algorithm. However, the main advantage of this method is the reduction of the edges in dense graphs, which can be used as a preprocessing step for other algorithms.

As we will later see, graphs in computer vision often have only $O(n)$ edges, therefore, this technique is of minor interest. Nevertheless, the techniques are interesting from a theoretical point of view and used in the algorithms later presented. In the early work of Benczúr and Karger only cuts were considered [5] but later they provided sampling methods which allowed the computations of approximate max flow [6]. In this section only the results are presented since a in depth discussion would be beyond the scope of this thesis.

Theorem 6 (Random Sampling [6])**.** *Given a weighted graph $G = (V, E, w)$ and a $\varepsilon > 0$, in time $O(m \log^3(n))$ one can construct a graph $\tilde{G} = (V, \tilde{E})$ such that*

1. *$\tilde{G}$ has $O(n \log(n) \varepsilon^{-2})$ edges*
2. *the value of every cut in $\tilde{G}$ is $(1 \pm \varepsilon)$ the value of the cut in G*

For an unweighted graph the runtime is reduced to $O(m \log^2(n))$.

The strategy to achieve such a graph $\tilde{G}$ is to assign 'important' edges a higher probability where edges crossing a small cut are considered to be important. This is somehow reasonable since the cut value of a small cuts is more sensitive with respect to the sampled edges.

Sampling in residual graphs

This algorithm is a consequence of the results obtained from the random sampling procedures. The main idea is to search for augmenting paths in the residual network but instead of searching in the whole network the search is performed only on a subset of the edges. This subset is the same as the one obtained by the Random Sampling methods. Since these probabilities prefer edges crossing small cuts the number of augmenting paths will decrease faster. This allows the construction of a max-flow algorithm with runtime $\tilde{O}(m + nF)$

3.1.3. Capacity Scaling

In this section a method is presented to bound the ratio of the capacities in a graph and still get a valid $(1-\varepsilon)$-approximate flow. One method is to rescale the capacities in a

way such that the ratio between the minimal and maximal capacity is less than $2m^2/\varepsilon$. In this modified graph a $(1 - \varepsilon/2)$-approximate max flow is equal to a $(1 - \varepsilon)$-approximate flow in the original graph.

The following method is presented in the paper of Christiano [19] where bounded capacities are an important requirement.

The basic idea is to calculate a maximum bottleneck (which can be computed quite fast in $O(m + n \log n)$ time [56]). Afterwards, large capacities are bounded and (irrelevantly) small capacities are removed.

For a given graph $G = (V, E)$ with weights w, the problem of finding the maximum bottleneck is given by:

$$\max_{P:\ s-t \text{ path}} \quad \min_{e \in P} \quad w(e)$$

This can be solved by simply applying Dijkstra's shortest-path algorithm with some small changes:

- the start vertex is s
- the 'distance' of a vertex v is the minimal weight of all paths form s to v
- this distance is maximized
- for a marked vertex u, unmarked vertex v: $d(v) = \max\left(d(v), \min(d(u), w(u, v))\right)$

Dijkstra's algorithm, if used with Fibonacci Heaps, has a worst case runtime of $O(m + n \log n)$ [25].

Now let B denote this maximum bottleneck value. It follows immediately that the maximum flow is at most mB and at least B, therefore, capacity larger than mB can be reduced to capacity mB. On the other hand if edges with capacity less than $\varepsilon B/2m$ are removed the change in the maximum flow is at most $\varepsilon B/2$. In this modified graph, the maximal possible ratio of the capacities is $2m^2/\varepsilon$ as desired.

Let F^* be the optimal flow value in the original graph. The optimal value in the modified graph is greater than $F^* - \varepsilon B/2 \geq F^* - \varepsilon F^*/2 = F^*(1 - \varepsilon/2)$. If f is a $(1 - \varepsilon/2)$-approximation in the modified graph, and F its flow value. Then it holds that:

$$\begin{aligned} F &\geq (1 - \varepsilon/2)F^*(1 - \varepsilon/2) \\ &\geq F^* - 2\varepsilon/2 F^* + \varepsilon^2/4 F^* \geq (1 - \varepsilon)F^* \end{aligned}$$

which indicates that f is a $(1 - \varepsilon)$-approximation in the original graph.

Furthermore, the optimal value of the cut/flow lies in the interval $[B, mB]$, therefore, only $O(\log(m))$ binary search steps have to be performed.

3.2. Algorithm of Christiano et al.

In the original paper of Christiano et al. [19] three algorithm are presented. The first algorithm is a $(1 - \varepsilon)$-approximation of the max-flow problem, this algorithm has a

runtime of $\tilde{O}(m^{4/3}\varepsilon^{-3})$. The second algorithm is a slight, but efficient, modification of the first algorithm and improves the time complexity to $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$. The third algorithm is a min-cut approximation algorithm with runtime $\tilde{O}(m + n^{4/3}\varepsilon^{-8/3})$. Since the algorithms are based on electrical flows they only work for undirected graphs. In the paper they only consider integer capacities, but this should not be much of a problem since one simply can rescale them (as long as they are rational). As we will see later these algorithms have a very good asymptotic time complexity, unfortunately the constants are quite high and the dependence on the parameter ε is quite drastic.

The central definition of the algorithm is the notion of congestion:

Definition 28. (congestion)
For a graph $G = (V, E)$, with capacities u and a s-t flow f, the congestion of an edge $e \in E$ is defined as

$$\text{cong}_f(e) := |f| / u_e$$

It is clear that the flow f respects the capacity constraints iff the congestion on every edge is at most 1. The main idea of the flow algorithm is to find resistances, such that for the corresponding electrical flow, the congestions are at most 1.

As described at the beginning of this chapter, all three algorithms have the form of an oracle. They answer the question if a feasible flow, with a certain flow, value exists. If such a flow exists the algorithm will return this flow, otherwise signalize that there is no such flow (i.e. return `fail`)
With this oracle one can perform binary search in order to compute an optimal flow or cut. In order to perform binary search, the technique presented in 3.1.3, is used to lower and upper bound the maximum flow value. As demonstrated the maximum bottleneck B can serve as a lower bound and m times this value is a upper bound. This reduces the 'search interval' to $[B, mB]$ and since integer capacities are assumed, the number of steps to find the optimum is logarithmic.

3.2.1. Maximum Flows

The simple $\tilde{O}(m^{4/3}\varepsilon^{-3})$ algorithm

As mentioned before the algorithm tries to adjust the resistances for a flow of a given flow value. This is accomplished by adopting some ideas from the Multiplicative Weights Update (MWU) method presented by Arora et al. [1]. The MWU method can be used to approximately calculate a solution $x \in \mathbb{R}^n$ such that $x \in P$ and $Ax > b$ for a convex set P, a matrix A and a vector b. Furthermore, a oracle, that returns a $x \in P$ such that $c^t x \geq d$ for a vector c and a constant d, is required. If no such x exists the oracle should return `FAIL`. This oracle is now queried multiple times for different c and d in order to get a solution for the original problem. The runtime of this method depends on the runtime of the oracle, but since the oracle only has to consider *one* constraint on can probably solve this problem a lot faster. A overview over this technique is given in the appendix (section A.1).

While the main structure of the algorithm of Christiano et al. is very close to the structure of the MWU-framework, the requirements for the oracle are quite different.

Oracle

The main part of the algorithm of Christiano et al. is the construction of a so called (ε, ρ)-oracle. This oracle produces a flow where the congestion of every edge can be greater than 1, but the congestion still is bounded by ρ. On the other hand not too many edges are allowed to have a high congestion. Formally this is expressed in the following definition:

Definition 29. ((ε, ρ) -Oracle)
Given $G = (V, E)$, with capacities u. Let F^* be the flow value of the maximum flow. Furthermore, let $w \in \mathbb{R}^{|E|}, w \geq 1$ be some weight vector on the edges and $\varepsilon, \rho, F > 0$ are given constants. Then a (ε, ρ)-oracle is a algorithm with the following return values:

1. if $F \leq F^*$, return flow f with
   (i) $|f| = F$
   (ii) $\sum_e w_w \operatorname{cong}_f(e) \leq (1 + \varepsilon) \sum_e w_e$
   (iii) $\max_e \operatorname{cong}_f(e) \leq \rho$
2. if $F > F^*$ fail or a flow with conditions (i), (ii) and (iii).

If compared with the requirements for the oracles of section A.1.2 (i) is related to $x \in P$, property (ii) is loosely based on $c^t x \geq d$ and as we will later see (iii) kind of bounds the penalties.

A $(\varepsilon, 3\sqrt{m/\varepsilon})$-oracle can be constructed by just setting

$$r_e = \frac{1}{u_e^2}\left(w_e + \frac{|w|_1 \varepsilon}{3m}\right)$$

and calculating a $(1 + 3\varepsilon)$-approximation of the electrical flow with flow value F. The pseudocode is given in algorithm 1, and the the runtime is given by the runtime of the electrical flow calculation, i.e. the solver of the Laplacian system (see section 3.1.1).
For a proof why this algorithm really fulfills the requirements of an (ε, ρ)-oracle see [19].

The runtime of this oracle is dominated by the calculation of the electrical flow. Recall that the ratio between maximum and minimum resistance is polynomial bounded and with Theorem 5 it follows that the oracle has a runtime of $\tilde{O}(m \log(\varepsilon^{-1}))$.

Algorithm 1 Christiano $(\varepsilon, 3\sqrt{m/\varepsilon})$ Oracle

Input: $G = (V, E)$, capacities u, target flow value F, weight vector w
Output: flow $\bar{f}$ or false indicating $F > F^*$

$r_e \leftarrow 1/u_e^2 \left(w_e + \frac{\varepsilon \|w\|_1}{3m}\right) \qquad \forall e \in E$
calculate an $\varepsilon/3$-approximate electrical flow $\tilde{f}$ with resitances r and flow value F
if $\mathcal{E}_r(\tilde{f}) > (1+\varepsilon)\|w\|_1$ **then**
    return `false`
else
    return $\tilde{f}$

Algorithm 2 Christiano MWU iterations

Input: $G = (V, E)$, capacities u, target flow value F and $(\varepsilon, 3\sqrt{m/\varepsilon})$-Oracle O
Output: flow $\bar{f}$ or `false` indicating $F > F^*$

$w_0 \leftarrow 1$, $N \leftarrow \frac{2\rho \ln m}{\varepsilon^2}$
for $i = 1 \cdots N$ **do**
    $f^i = O(G, u, F, w^i)$
    if O returned false **then**
        return `false`
    else
        $w_e^i \leftarrow w_e^{i-1}(1 + \frac{\varepsilon}{\rho} \operatorname{cong}_{f^i}(e)) \ \forall e \in E$ ▷ MWU step
return $(1-\varepsilon)^2/(1+\varepsilon)N \sum_i f^i$

MWU step

Using the $(\varepsilon, 3\sqrt{m/\varepsilon})$ oracle the rest of the algorithm is now straight forward and given in Algorithm 2.

The main loop of this algorithm is exactly the same as in algorithm 14, where the penalties are $\text{cong}_f(e)/\rho$. The convergence of this algorithm is given by the following theorem:

Theorem 7 ((Christiano et al. [19]) Appr. Maximum Flow via MWU). *Given $0 < \varepsilon < 1/2$, $\rho > 0$ and a (ε, ρ) oracle with running time $T(m, 1/\varepsilon, U)$ Algorithm 2 computes a $(1 - O(\varepsilon))$- approximate maximum flow in an capacitated, undirected graph with running time $\tilde{O}(\rho/\varepsilon^2 T(m, 1/\varepsilon, U))$.*

For a proof see again [19].

Combining this theorem with the oracle given by algorithm 1, which has runtime $\tilde{O}(m \log \varepsilon^{-1})$, yields the claimed $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$ max flow approximation algorithm. The exact formulation of the theorem describing the runtime is:

Theorem 8 (Christiano [19]). *For any $0 < \varepsilon < 1/2$ the maximum flow problem can be $(1 - O(\varepsilon))$-approximated in $\tilde{O}(m^{3/2}\varepsilon^{-5/2})$ time.*

The improved $\tilde{O}(mn^{1/3}\varepsilon^{-11/3})$ algorithm

Theorem 7 indicates that the value of ρ plays an important part in the runtime. Unfortunately $\rho \in \Theta(\sqrt{m})$ has to be chosen. This is illustrated by the example given in Figure 2.1. In this example one edge got half of the flow and the rest of the flow was distributed equally on the rest of the edges/paths. The main observation now is the following. If in the example the highly congested edge $e = (s, t)$ is removed the flow on the remaining edges is only doubled for each edge, therefore, it does not change severely.

This leads to a modified oracle which simply removes a edge permanently if this edge is highly congested, see algorithm 3. The whole algorithm is given by algorithm 4. Note that the main difference is the additional set H which covers the forbidden edges. A edge is forbidden if its congestion exceeds $8m^{1/3}\ln^{1/3}(m)\varepsilon^{-1}$. The rests is basically unchanged (besides the fact that if a edge is removed the oracle calculation is restarted). The runtime is improved since ρ is smaller, therefore, the number of electrical flow computations is reduced.

Theorem 9 (Christiano [19]). *For any $0 < \varepsilon < 1/2$, if $\leq F^*$ then Algorithm 4 returns a feasible $s - t$ flow of value $(1 - O(\varepsilon))F$ in time $\tilde{O}(m^{4/3}\varepsilon^{-3})$.*

The runtime in this theorem is also the total runtime for the maximum flow problem since $O(\log(m))$ binary search steps are performed which are already included by the $\tilde{O}$-notation.

Algorithm 3 Christiano modified oracle

Input: $G = (V, E)$, capacities u, target flow value F, weights w and $H \subset E$ the set of forbidden edges
Output: flow $\tilde{f}$, $H \subset E$ or false indicating $F > F^*$

$\rho \leftarrow \frac{8m^{1/3} \ln^{1/3} m}{\varepsilon}$ ▷ added
$r_e \leftarrow \frac{1}{u_e^2} \left(w_e + \frac{\varepsilon |w|_1}{3m} \right) \qquad \forall e \in E$
$G_H \leftarrow (V, E \setminus H)$
$\tilde{f} \leftarrow (1 + \varepsilon/3)$-approximate electrical flow in G_H with resitances r and flow value F
if $\mathcal{E}_r(\tilde{f}) > (1 + \varepsilon) |w|_1$ or s,t are not connected **then**
    return false
if $\exists e : \text{cong}_{\tilde{f}}(e) > \rho$ **then**
    $H \leftarrow H \cup \{e\}$
    return $O_m(G, w, F, H)$
return $(\tilde{f}, H)$

Algorithm 4 Christiano MWU modified

Input: $G = (V, E)$, capacities u, target flow value F and a modified oracle O_m
Output: flow $\tilde{f}$ or `false` indicating $F > F^*$

$w_0 \leftarrow 1$, $\rho \leftarrow \frac{8m^{1/3} \ln^{1/3} m}{\varepsilon}$, $N \leftarrow \frac{2\rho \ln m}{\varepsilon^2}$, $H \leftarrow \emptyset$
for $i = 1 \dots N$ **do**
    $f^i, H' = O_m(G, w^i, F, H)$
    if O returned false **then**
        return `false`
    else
        $H \leftarrow H'$
        $w_e^i \leftarrow w_e^{i-1}(1 + \frac{\varepsilon}{\rho} \text{cong}_{f^i}(e)) \; \forall e \in E$ ▷ MWU step
return $\frac{(1-\varepsilon)^2}{(1+\varepsilon)N} \sum_i f^i$

3.2.2. Minimum Cuts

Similar to the max-flow algorithm presented previously, Christiano et al. also present a min-cut algorithm in their paper. The max-flow-min-cut theorem states that the flow value of the maximum flow is equal to the value of the minimum cut. This implies that all the edges in the minimum cut have congestion 1. The cut algorithm of Christiano builds on the observation that in small cuts the edges are more likely to be high congested compared to the edges in large cuts. Of course one has to consider that edges of a small cut also lie in large cuts. Similar to the maximum flow algorithms of the previous section the min-cut algorithm increases the weights according to the congestion, where high congestion means more weight in the next iteration. At the end most of the weight is concentrated on edges lying in small cuts (with respect to the total weight).

In order to define a measurement, indicating how much weight is already concentrated on small cuts, the properties of the effective conductance/resistance (see Definition 27) are used.

Given some potentials $\phi \in [0,1]^n$ with $\phi_s = 1$, $\phi_t = 0$ then one can obtain a valid cut by thresholding, this is sometimes called 'sweep cut'. If thresholding at a random value, the expected value of the resulting cut is $\sum_{e=(u,v)\in E} |\phi_u - \phi_v| \, w_e$, see Theorem 17 in Section 5.2.3 for a discussion of this property. For this value Christiano et al. provided the following bound:

Lemma 2. *Given a graph $G = (V, E)$ and resistances r. Let $\phi \in [0,1]^n$ the potentials of the electrical flow, then:*

$$\sum_{e\in E} \phi_e u_e \leq \sqrt{\frac{\sum_{e\in E} u_e^2 r_e}{\mathrm{R}_{\mathit{eff}}(r)}}$$

If the ϕ are the potentials of an approximate electrical flow calculated according to Theorem 5 with $\delta < 1/3$ (and scaled to lie in $[0,1]^n$), then:

$$\sum_{e\in E} \phi_e u_e \leq (1+2\delta)\sqrt{\textstyle\sum_{e\in E} u_e^2 r_e / \mathrm{R}_{\mathit{eff}}(r)}$$

Proof. recall that $\mathrm{C}_{\mathrm{eff}}(r) = \sum_{e=(u,v)\in E} |\phi_u - \phi_v| / r_e$ and that the effective resistance $\mathrm{R}_{\mathrm{eff}}(r)$ is the inverse of $\mathrm{C}_{\mathrm{eff}}(r)$.

$$\sum_{e\in E} \phi_e u_e \leq \sqrt{\sum_{e=(u,v)\in E} \frac{|\phi_u - \phi_v|}{r_e} \sum_e u_e^2 r_e} = \sqrt{\frac{\sum_e u_e^2 r_2}{\mathrm{R}_{\mathrm{eff}}(r)}}$$

The first inequality is the Cauchy-Schwarz inequality. The second part of the theorem follows from Theorem 5 (*i*) and (*iii*). □

In the rest of the analysis of the algorithm Christiano et al. showed that after $N = 5\varepsilon^{-8/3} m^{1/3} \ln m$ iterations the resistances fulfill

$$\mathrm{R}_{\mathrm{eff}}(r) \geq (1 - 7\varepsilon)\frac{\mu}{F^2}$$

Combining this equation with Lemma 2 the following theorem follows:

Theorem 10 (Christiano et al. [19]). *Given $\varepsilon < 1/7$ and a $F \geq F^*$, Algorithm 5 returns a cut with value less than $F/(1-7\varepsilon)$ and has time complexity $\tilde{O}(m^{4/3}\varepsilon^{-8/3})$.*

Which indicates that there is a cut smaller than $\frac{1}{1-7\varepsilon}F$ for some $\varepsilon < 1/7$ this clearly yields and $(1+O(\varepsilon))$-approximate cut.

Algorithm 5 Christiano cut algorithm

Input: graph $G = (V, E)$, capacities u, target flow value F
Output: A cut $S \subset V$ with value $\leq (1+\varepsilon)F$ or `fail` if no such cut exist

1: $w_0 \leftarrow 1$, $\rho \leftarrow 3m^{1/3}\varepsilon^{-2/3}$, $N \leftarrow 5\varepsilon^{-8/3}m^{1/3}\ln(m)$, $\delta \leftarrow \varepsilon^2$
2: **for** $i := 1, \ldots, N$ **do**
3: $\quad r^{i-1} = w^{i-1}/u_e^2$
4: $\quad$ calculate a $(1+\delta)$ approximate flow $\tilde{f}^{i-1}$ and potentials ϕ
5: $\quad w_e^i = w_e^{i-1} + \frac{\varepsilon}{\rho}\operatorname{cong}_{\tilde{f}^{i-1}}(e)w_e^{i-1} + \frac{\varepsilon^2}{m\rho}|w^{i-1}|$
6: $\quad$ scale ϕ s.t. $\phi_s = 1$, $\phi_t = 0$
7: $\quad$ chose a $x \in (0,1)$ uniformly
8: $\quad S_x \leftarrow \phi > x$ ▷ sweep cut
9: $\quad$ **if** $val(S_x) \leq \frac{F}{(1-7\varepsilon)}$ **then**
10: $\quad\quad$ **return** S_x
11: **return** `fail`

In Section 5.2.3 another sweep cut algorithm, calculating the optimal threshold, is discussed (Algorithm 9). This optimal algorithm could be used instead of random thresholding in line 8 of Algorithm 5 without affecting the (asymptotic) runtime. On the other hand the theoretical properties do not change ether.

3.2.3. Discussion and Experiments

The previous algorithms are only of theoretical interest. For a practical use they are way to slow since the constants, hidden in the O-Notation of the runtime, are too big. This is illustrated by the following observation: if for a certain flow value a feasible flow exists, the flow algorithm requires $2\rho\ln(m)\varepsilon^{-2}$ oracle calls where $\rho = 8m^{1/3}\ln^{1/3}(m)\varepsilon^{-1}$. In total

$$N = 16m^{1/3}(\ln(m))^{4/3}\varepsilon^{-3}$$

oracle calls. Suppose that $\varepsilon = 0.1$ which is a conservative choice (the solution can deviate by 10%) and set $m = 1000$ (which is a very small image as we will discuss later) more than 2000000 oracle calls (i.e. electrical flow computations) have to be performed. In this calculation the ε^{-3} term is the critical one since it accounts for most of the 2000000 oracle calls. Of course if m gets larger the influence of this term decreases (for $m = 10000$ about 6500000 calls have to be made) but as seen later the primal-dual Algorithm needs (in practice) less iteration and every iterations is cheaper then a electrical flow computation.

Also note that these calculations do not yet include the $O(\log(m))$ binary search steps. While not every search step will require the total amount of N oracle calls and the upper bound for the maximum flow, m times the maximum bottleneck, is very crude. But in practice this sure has an impact if an algorithm has to run multiple times.

In the cut algorithm the number of oracle call is a little bit less:

$$N = 5\varepsilon^{-8/3} m^{1/3} \ln m$$

Still, around 16000 calls are necessary for $m = 1000$.

Experiments

A Matlab implementation of the cut algorithm of Christiano was run with different instances. The results were performed on a Intel Core i7-4820K CPU (4 cores + hyper-threading, 3.7Ghz) running Ubuntu 14.04LTS (64 bit) and Matlab 2013a. The input was a simple segmentation task of an image with resolutions 16x16, 32x32 and 64x64 pixels. For the simple segmentation task every pixel was assigned to a vertex, this vertex was connected to its 4 neighbours (or less on the border of the image). Furthermore, every vertex was connected to the source and the sink. The weights were calculated from the gray values of the pixels. More details about how these values are obtained will be given later in section 4.1.

Figure 3.1.: Result of the Algorithm of Christiano, difference to the real solution

The value of ε was 0.1, i.e. an error of 10 percent was acceptable. To the best of our knowledge no implementations of a near linear time solver exists. Therefore, for most of the computations the electrical flows were commutated with the built-in preconditioned gradient descent method (PCG), as preconditioner the incomplete LU decomposition (ilu) was used. Compared to the standard Matlab '\' operator or `mldivide`, this method turned out to be much more robust against numerical issues (the underlying linear systems are very ill conditioned). Additional the Lean Algebraic Multigrid (LAMG)[41], developed by Livne and Brandt was used. Unfortunately, probably because of the high overhead, this solver was slower the PCG.

In table 3.1 the basic characteristics are given.

resolution	16x16	32x32	64x64
n	258	1026	4098
m	965	3868	15612
wrong labels	0	0	25
cut value	6440	15609	61300 (gt: 61258)
ρ	137.6037	218.5831	348.0226
N	157605	300934	560072
$\max w_i$	7.728×10^{16}	1.358×10^{20}	2.886×10^{23}
$\max_w \sum_i w_i$	1.175×10^{19}	4.091×10^{22}	3.365×10^{26}
time	$\sim$ 27 min	$\sim$ 177 min	$\sim$ 20h

Table 3.1.: Run time and other characteristics for a simple example

The most noticeable fact is probably the very high runtime. This of course is due to the high number of N, i.e. the maximum number of iterations. The approximation parameter ε was chosen quite conservative but still the two smaller problem instances were solved exactly and the third one was only slightly away from the optimum (the relative error is within 10^{-3}).

Because of the already very high runtimes no calculations for larger problem instances were carried out.

Even for these small instances the weights w in the algorithm grew quite large. Matlab uses the double precision floating point format, specified by the standard IEEE 754, for the calculations. The precision of the significand is 53 bits, this means that values larger than $2^{53} \approx 10^{16}$ may not be represented exactly in this format. As shown by the experiments even in the smallest problem instance some values are exceeding this number. For calculations carried out in single precision, only values up to $2^{23} \approx 10^{7.2}$ can be represented exactly. If 10^{26} is represented as a single precision number the difference to the next closest single precision number can be as large as 10^{18}. While a overflow is not imminent (the largest value a single precision floating point value can represent is around 10^{38}) the calculations would be far from being exact. Since, nowadays computations are often performed on the GPUs, most of them optimized for single precision calculations, these large values may could lead to problems.

Nevertheless, the asymptotic runtime of the algorithm is outstanding and justifies this kind of algorithms.

3.3. Algorithm of Lee et al.

3.3.1. Preliminaries

In this section two algorithm are presented. The first one is a max-flow algorithm and the second one a min-cut algorithm. Both algorithm appear in the paper of Lee et al. [42], hence the name. Both algorithm are somehow special in the sense that their (main) result is valid only for uncapacitated graphs, i.e. graphs with capacity 1 for each edge.

The runtime in the general case is much worse and can not compete with the other algorithms presented in this thesis. Nevertheless, the algorithms provide some interesting interpretation about max-flow problems.

The main difference to the algorithm of Christiano (see Section 3.2) is that this algorithm does not punish edges according to their overflow (and thus does not use the MWU framework) but instead tries reroute the overflow. This rerouting is accomplished using electrical flows.

The algorithm basically consists of Nesterov's Accelerated Gradient Descent Method and a 'error correction procedure' to ensure that the output satisfies the capacity constraints.

The algorithm uses the terminology of cycle and cut spaces and in order to justify these names we have to link vector spaces with graphs.

For a given graph G the most basic vector spaces are the s.c. *vertex space* and *edge space*. The vertex space denoted by C_V is simply is the space of all functions from the vertex set $V(G)$ into the $\mathbb{C}$. Similar the edge space C_E is the space of all function from $E(G)$ into $\mathbb{C}$. The dimension of the C_V is $|V| = n$ and the dimension of C_E is $|E| = m$. A element $f \in C_V$ can be written as a vector $(f(v_1), \ldots, f(v_n))$ and analog for the elements of C_E.

Now let $G = (V, E)$ be a directed graph and S of G. The cut vector $y_S \in C_E$ is defined as follows:

$$(y_S)_i := \begin{cases} 1 & \text{if } e_i = (u, v) \text{ with } u \in S, v \notin S \\ -1 & \text{if } e_i = (u, v) \text{ with } u \notin S, v \in S \\ 0 & \text{if } e_i \notin C(S) \end{cases}$$

The cut space of G denoted by $\mathfrak{C}(G)$ is the space spanned by all cut vectors for all possible cuts of G.

Analog to the cut space one can define the cycle space by creating the cycle vector y_P of a cycle $P = p_1 p_2 \ldots p_k$

$$(y_P)_i := \begin{cases} 1 & \text{if } \exists j | e_i = (p_j, p_{j+1}) \\ -1 & \text{if } \exists j | e_i = (p_{j+1}, p_j) \\ 0 & \text{else} \end{cases}$$

Now the cycle space of G is the space spanned by all the cycles and is denoted by $\mathfrak{C}^\perp(G)$. If it is clear from context, which graph is meant, the subscript is dropped and $\mathfrak{C}$ and $\mathfrak{C}^\perp$ is written.

It is easy to see that $\mathfrak{C}$ and $\mathfrak{C}^\perp$ are orthogonal since for arbitrary cut vectors y_s and cycle vectors y_P, $\langle y_P, y_S \rangle$ is the difference of the number of edges from S to $V \setminus S$ and the number of edges from $V \setminus S$ to S in the circle P. Therefore, $\langle y_P, y_S \rangle = 0$, indicating that the vectors are orthogonal. Furthermore, it is a well known fact the sum of $\mathfrak{C}$ and $\mathfrak{C}^\perp$ spans the whole cutspace (see [7] for a proof).

A central result in algebraic graph theory is that the cut space and the cycle space are

described by the incidence matrix of the graph:

$$\mathfrak{C} = \mathrm{Im}(B)$$
$$\mathfrak{C}^{\perp} = \ker(B^t)$$

again see[7] for a proof. The following algorithms makes heavy use of (orthogonal) projections onto the space $\mathfrak{C}^{\perp}$. Therefore, let Π denote this orthogonal projection, the projection matrix is given by:

$$\Pi = BL^{+}B^t$$

It is easy to verify that this matrix is a projection matrix and that the projection is orthogonal. Analog the projection onto the cycle space is denoted by $\Pi^{\perp}$ and, by the simple rules of projection matrices, is given by

$$\Pi^{\perp} = I - \Pi = I - BL^{+}B^t$$

3.3.2. Maximum Flows

Finding a feasible flow

Recall that the flow preservation can be expressed as $B^t f = F\chi$ for a flow f, incidence Matrix B, flow value F and χ the vector identifying s and t. Therefore, the set of feasible s-t flows with flow value F, denoted by $\mathfrak{F}_{st,F}$, is given by:

$$\begin{aligned}\mathfrak{F}_{st,F} :&= \left\{f : B^t f = F\chi\right\}\\ &= \left\{f : BL^{+}B^t f = FBL^{+}\chi\right\}\\ &= \{f : \Pi f = F f_{st}\}\end{aligned}$$

where $f_{st} := BL^{+}\chi$ is the unit electrical flow from s to t.

By further rewriting $\mathfrak{F}_{st,F}$ one gets:

$$\begin{aligned}\mathfrak{F}_{st,F} &= \{f : \Pi f = F f_{st}\}\\ &= F f_{st} + \{f : \Pi f = 0\} \quad = F f_{st} + \left\{f : BL^{+}B^t f = 0\right\}\\ &= F f_{st} + \left\{f : B^t f = 0\right\} = F f_{st} + \ker(B^t)\\ &= F f_{st} + \mathfrak{C}^{\perp}\end{aligned}$$

The last equation describes a translation of the cyclespace by the electrical s-t flow with current F. Furthermore, this electrical flow is given by the projection:

$$F f_{st} = \mathcal{P}_{\mathfrak{F}_{st,F}}(0)$$

See figure 3.2 for a graphical illustration.

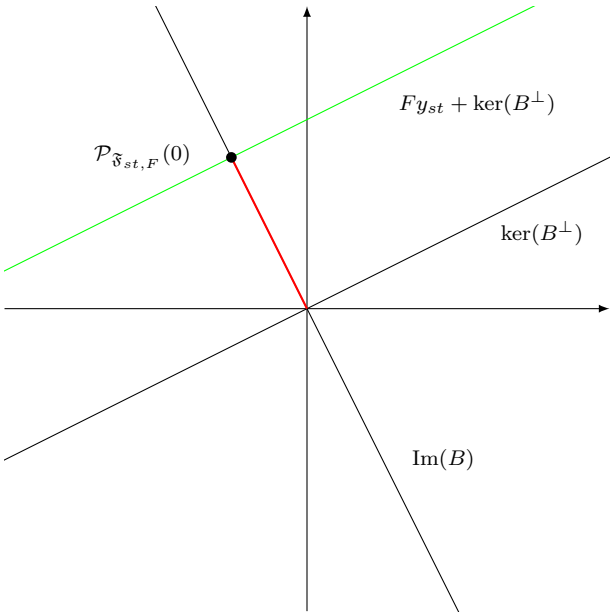


Figure 3.2.: Illustrating the projections of the feasible flows for the 1-dimensional case

As mentioned earlier the main results are only valid for uncapacitated flows (i.e. all capacities are 1) and for now only this case will be considered. Therefore, the flow on every vertex has to lie in the interval $[-1, 1]$, and every feasible flow f must be an element of the unit sphere of the infinity norm (denoted by B_∞^m) and an element of the set of feasible flows $\mathfrak{F}_{st,F}$. Therefore, in order to answer the question whether a flow with flow value F exists one has to find a $f \in B_\infty^m \cap \mathfrak{F}_{st,F}$. This can by achieved by minimizing $\|f\|_\infty$ inside $\mathfrak{F}_{st,F}$ but since this problem is not differentiable one can instead minimize the distance to the projected point, i.e.

$$\min_{f \in \mathfrak{F}_{st,F}} \underbrace{\frac{1}{2}\left\|f - \mathcal{P}_{B_\infty^m}(f)\right\|_2^2}_{\phi(f)} \tag{3.2}$$

This problem is convex, has Lipschitz constant 1 and the gradient is given by $\nabla\phi(f) = f - \mathcal{P}_{B_\infty^m}(f)$.

A projection onto a box (located at the center and aligned to the axis) is a achieved by truncating the coordinates.

Note that the gradient is given by:

$$(\nabla\phi(f))_i = \begin{cases} 0 & f_i < c_i = 1 \\ f_i - c_i & \text{else} \end{cases}$$

Therefore, the gradient is exactly the amount of 'overflow', i.e. the amount of flow exceeding the constraints.

The gradient for f constrained to $\mathfrak{F}_{st,F}$ is given by:

$$\nabla\phi_{\mathfrak{F}_{st,F}}(f) = \Pi^\perp \nabla\phi(f) = (I - BL^+B^t)\nabla\phi(f)$$

Gradient descent steps for the objective function g are of the form '$x^{i+1} = x^i - \nabla g(x^i)$'. In this case this would be:

$$f^{i+1} = f^i - \nabla\phi(f^i) + BL^+B^t\nabla\phi(f^i)$$

Which can be interpreted as subtracting the overflow and adding an electrical flow were at each vertex $B^t\nabla\phi(f^i)$ current is induced, i.e. $c_{ext} = B^t\nabla\phi(f^i)$

Now any gradient descent method can be used to minimize the problem given in (3.2). Let f^* be a solution of (3.2) then this flow will fulfill the flow preservation constraints. The initial question, whether a flow with value F exists, can be answered by checking if f^* additionally fulfills the capacity constraints.

Since only a $(1-\varepsilon)$-approximation is required one may stop the gradient descent scheme before it is fully converged. One drawback of this approach is that in this case the approximate solution may not be feasible and violates the capacity constraints. In order to get a feasible flow a s.c. 'drain procedure' is required to reroute this overflow.

Drain procedure

Let $G = (V, E)$ be a directed graph and suppose that f is a approximative solution of (3.2) yielded by some gradient descent like method. The task is to convert, or round, this flow to a feasible s-t flow.

First of all G is converted into a directed graph, such that all the edges have nonnegative flow. Recall that actually only nonnegative flows are considered. Now if the flow value at a edge is negative, this edge is considered orientated in the opposite direction, i.e. by reversing the orientation of an edges the sign of its flow changes. Therefore, by reversing all edges with negative flow, the resulting graph has nonnegative flow.

Let $D = \{(s_1, t_1), \cdots, (s_k, t_k)\}$ be the set of congested edges. Now a new vertex d is added to the graph. For each congested edge $(s_i, t_i) \in D$ two new edges (s_i, d) and (d, t_i) are created. A new flow $\bar{f}$ is created on this new graph where the flow on each original edge e_i is given by $\bar{f}(e_i) = \min(f(e_i), c(e_i))$ and the flow for the new edges are given by $\bar{f}(s_i, d) = \bar{f}(d, t_i) = f(s_i, t_i) - c(s_i, t_i)$. This augmented flow $\bar{f}$ is a feasible flow (the capacity on the newly created edges are set to infinity). In order to get a s-t flow in the original graph G one has to 'drain' the flow going through the newly created vertex d. This is achieved by iteratively finding paths from d to t along the edges with non zero flow value. Searching for the minimum flow along this path and subtracting this value from all the edges on the path. Analog for paths from s to d. If there is no path from s to d (or d to t) then no flow is passing through d. By removing d (and its incident edges) one obtains a feasible s-t flow in G.

From the above description it is observed that it is crucial to perform the following steps very quickly: find a path from d to s and t, find the minimum flow value along this path and augment the flow along this path. In their paper Lee et al. used *Dynamic Trees*[58] also known as *Link/Cut trees* to obtain the desired $O(m\log(n))$ runtime of the drain procedure. This is formulated in the following Lemma

Lemma 3 (Lee et al. [42], Lemma 2 (Overflow Drainage)). *Suppose f is a (not necessarily feasible) s-t flow of value F. Then there is a feasible s-t flow f' of value at least $F - \sum_{e \in E} |\max(0, f(e) - c(e))|$. Moreover, there is an algorithm Drain which finds such an f' in $O(m \log(n))$.*

The max-flow algorithm

In order to solve optimization problem (3.2) the Nesterov's Accelerated Gradient Descent Method is used, since it converges faster than the standard gradient descent method. For a short outline of this method see Section A.2. This algorithm is only run for $2/\varepsilon \sqrt{m/F}$ iterations and the resulting solution is converted into a valid flow by the drain procedure.

Putting all the steps together this results in the following algorithm:

Algorithm 6 Lee et al. max-flow

Input: $G = (V, E)$, target flow value F, $\varepsilon > 0$
Output: A feasible flow (if there exists one)

$y_0 = FBL^+\chi$
$y_T = \text{NESTEROV}(\Pi^{\perp}\nabla\phi, 1, 0, \frac{2}{\varepsilon}\sqrt{\frac{m}{F}}, y_0)$
$f = \textsc{Drain}(y_T/(1+\varepsilon), F)$

Theorem 11 (Lee [42], Max Flow). *If a feasible flow with value F exists then algorithm 6 finds a feasible flow of value $(1 - 4\varepsilon)F$ after $O(\frac{1}{\varepsilon}\sqrt{\frac{m}{F}} m \log^2 n)$ time.*

See the original paper for a prove. In contrast to Christiano et al., Lee et al. never mentioned what happens when no feasible flow exists in their paper. The simplest approach would be to check if the result of the algorithm really is a flow with flow value $> (1 - \varepsilon)F$ and output `fail`. But from carefully reading the proof of the theorem one can see that if more than $2\varepsilon F$ edges are heavily congested (congestion $> \varepsilon$), the return value should be `fail`.

As mentioned at the beginning of this chapter binary search has to be performed in order to find a maximum flow.

If the queried flow value F is greater than $(\frac{m}{n\varepsilon})^{2/3}$ this algorithm has a runtime of $\tilde{O}(mn^{1/3}\varepsilon^{3/2})$. If the $F < (\frac{m}{n\varepsilon})^{2/3}$ then the algorithm of Karger-Levine introduced in Section 3.1.2, which has a good running time in graphs with small flow, is used. This algorithm has time complexity $\tilde{O}(m + nF)$.

Combining these two methods yields the following result:

Theorem 12. *A $(1 - O(\varepsilon))$-approximation of the maximum s-t flow problem can be computed in time $\tilde{O}(mn^{1/3}\varepsilon^{3/2})$.*

3.3.3. Minimum Cuts

The algorithm for the min-cut problem is quite similar to the max-flow algorithm. The main part is again Nesterov's Accelerated Gradient Descent Algorithm. The gradient of the objective function again contains a projection, therefore, the result again rely heavily on a fast SDD solver.

In order to use the same framework as in the flow algorithm one has to find a convenient description of the cut space. Now consider the following LP-formulation of the min-cut problem:

$$\min_{\substack{x\in\mathbb{R}^n \\ \langle x,\chi\rangle=1}} \|Bx\|_1 \tag{3.3}$$

See section 4.1 for a proof why this is a valid formulation of the graph cut problem. Since $\chi \in \mathrm{Im}(B^t) = \mathrm{Im}(L)$ it holds that $\chi = LL^+\chi = B^tBL^+\chi$, therefore, the inner product in equation (3.3) can be reformulated as:

$$\min_{\substack{x\in\mathbb{R}^n \\ \langle Bx,BL^+\chi\rangle=1}} \|Bx\|_1$$

by setting $f := Bx$, this is equivalent to:

$$\min_{f\in\mathrm{Im}(B)\cap\{\langle f,f_{st}\rangle=1\}} \|f\|_1$$

For the sake of readability let $\mathfrak{C}_{st} := \mathrm{Im}(B) \cap \{\langle f, f_{st}\rangle = 1\}$. This is a subspace of the cutspace $\mathfrak{C}$ more precisely the space of s-t cuts. In order to use the projected gradient descent method one can rewrite the constraint $f \in \mathfrak{C}_{st}$ as linear System $Af = b$. It is easy to check that the matrix

$$A := \Pi^{\perp} + \frac{f_{st}f_{st}^t}{\|f_{st}\|_2}$$

and the vector

$$b := \frac{f_{st}}{\|f_{st}\|_2}$$

fulfills the requirement.

The problem in (3.3) is non-differentiable, therefore, the smooth and convex approximation $l_\mu(f) := \sum_{i=1}^m \sqrt{f_i^2 + \mu^2}$ is used. The gradient of ∇l_μ is straight forward and can be computed in linear time, since

$$\frac{\partial}{\partial f_i} l_\mu(f) = \frac{f_i}{\sqrt{f_i^2 + \mu^2}}$$

The partial derivatives of ∇l_μ are given by

$$\frac{\partial^2}{\partial^2 f_i} l_\mu(f) = {\mu^2}/{(f_i^2+\mu^2)^{3/2}}$$

which clearly is bounded by μ^{-1} and therefore the Lipschitz constant is μ^{-1}. This relaxed objective function l_μ is now minimized by Nesterov's Accelerated Gradient Descent Method. As previously for the flow algorithms, after T iterations Nesterov's Method is stopped. At this point the L_1 norm of the resulting vector f^T will be less than $(1+\varepsilon)F$. The corresponding approximate cut vector x^T will of course not be binary. In order to get a binary solution this vector is rescaled such that $x_s^T = 1$ and $x_t^T = 0$ and thresholded at a random $\rho \in (0,1)$. Analog as for the min-cut algorithm of Christiano the optimal sweep cut algorithm, in order to calculate the optimal thresholding value, could be used (Algorithm 9). Again using this optimal algorithm would not influence the asymptotic runtime nor the theoretical properties.

Theorem 17 and the fact that only unweighted graphs are considered yields that the resulting cut is less than

$$\sum_{(u,v)\in E} \left| x_u^T - x_v^T \right| = \left\| Bx^T \right\|_1 = \|y\|_1 \leq (1+\varepsilon)F$$

Putting all these steps together gives the following theorem.

Theorem 13 (Lee et al. [42], Min Cut)**.** *If there exists a s-t cut with value F in $G = (V,E)$. Then algorithm 7 will find a cut of value at most $(1+\varepsilon)F$ in time*

$$O\left(\frac{1}{\varepsilon}\sqrt{\frac{m}{F}} m \log^2 n\right)$$

For the proof see again the original paper. Again Lee et al. never mentioned what to

Algorithm 7 The algorithm of Lee et al.. The check if the obtained cut really is valid does not appear in their paper(6)

Input: $G = (V,E)$, target flow value F, $\varepsilon > 0$
Output: A s-t cut (if there exists one), `False` otherwise

$f_0 = \frac{f_{st}}{\left\|f_{st}^2\right\|_2} = \frac{BL^+\chi}{\|BL^+\chi\|_2}$
$f_T = \text{NESTEROV}((\Pi - \frac{f_{st}f_{st}^t}{\|f_{st}\|_2})\nabla l_\mu, \frac{1}{\mu}, 0, \frac{4}{\varepsilon}\sqrt{\frac{2m}{F}})$
$x_T = L^+B^t f_T$
choose $\alpha \in (0,1)$ uniformly
$\phi = x_T > \alpha$ ▷ random thresholding
if $\|B^t\phi\|_1 > (1+\varepsilon)F$ **then**
  return `False`
return S

do if a cut with value F does not exist. But again one simply has to check if the value of the obtained cut is smaller than $(1+\varepsilon)F$. Now again one is able to perform a binary search in order to obtain a min cut.

Analog to the Max-Flow algorithm the algorithm of Karger-Levine is used if the maximal flow is very small ($F < \frac{m}{n\varepsilon^{2/3}}$). This yields the claimed runtime of $\tilde{O}(mn^{1/3}\varepsilon^{-2/3})$

Theorem 14. *A $(1+\varepsilon)$-approximation of the minimum s-t cut problem can be computed in time $\tilde{O}(mn^{1/3}\varepsilon^{-2/3})$.*

3.3.4. Discussion and Experiments

The restriction to uncapacitated graphs is quite a drawback of the two algorithms. Of course one could convert a edge with integer weight into multiple edges with weights one. This is of course at the expense of the runtime since the algorithm would become pseudo-polynomial. Nevertheless for small weights this could be possible, although the practical running time would again increased since the number of gradient descent iterations depends on m.

But as seen from table 3.2 the runtime for small problem instances is already very high, therefore, these 'improvement' are more of a theoretical interest.

Experiments

For the experiments the same problem instances as for the algorithm of Lee was used (section 3.2.3). In order to get a uncapacitated problem, edges with high weights were included and edges with low weights were dropped.

Table 3.2 shows the running time for different image resolutions. Notice that the algorithm always calculated the correct cut, this indicates that the number of gradient descent iterations may be an 'overkill' and one could stop earlier for such small instances.

resolution	16x16	32x32	64x64	128x128	256x256
n	258	1026	4098	16386	65538
m	714	2377	10155	42661	179492
cut value	15	15	70	266	1404
wrong labels	0	0	0	0	32
time	$\sim$ 1 min	$\sim$ 3 min	$\sim$ 9 min	$\sim$ 40 min	$\sim$ 167 min

Table 3.2.: Running time and other characteristics for a simple example

Despite all these drawbacks the algorithm offers some very interesting theoretical insights and clearly is 'a new approach'. Same as the algorithm of Christiano et al. the asymptotic running time of the algorithms are clearly outstanding.

Part II.

Primal Dual Algorithms

4. Basic Definitions and Properties

In this chapter the graph cut problem is now formulated as a continuous convex optimization problem. The first section will cover different problem formulations and afterwards the essential definitions needed for the convex optimization algorithms are given.

4.1. Graph Cut Formulations

4.1.1. The standard graph cut

In this section some different but equivalent formulations of graph cut will be discussed. Starting with the most basic definition, which unfortunately are very abstract and not very practical for algorithmic purposes, some more practical formulations will be derived.

The most basic definition was already given by Problem (P_c) and was

$$\min_{\substack{\emptyset \neq S \subset V \\ s \in S, t \notin S}} \sum_{(u,v) \in C(S)} w(u,v)$$

where $C(S)$ was the cut set. Using the indicator function one can slightly reformulate the sum as

$$\min_{\substack{\emptyset \neq S \subset V \\ s \in S, t \notin S}} \sum_{e=(u,v) \in E} \mathbf{1}_{C(S)}(e) w(u,v) \tag{4.1}$$

where the indicator function $\mathbf{1}_C(e)$ is one if e is in the cut set and zero else.

Of course the indicator function can also be used for vertices and the set S. For a edge $e = (u,v) \in E$ this allows the reformulation of $\mathbf{1}_C(e)$ as $\mathbf{1}_C(e) = |\mathbf{1}_S(u) - \mathbf{1}_S(v)|$. Now lets fix a cut S and define the vector $x \in \{0,1\}^{|V|}$ as the incidence vector of the vertices belonging to that cut, i.e. for a vertex $u \in V$ let $x_u := \mathbf{1}_{C(S)}(u)$.

Plugging this into (4.1) on gets

$$\min_{\substack{x \in \{0,1\}^n \\ x_s = 1, x_t = 0}} \sum_{(u,v) \in E} w(u,v) \, |x_u - x_v|$$

assuming that the weights are nonnegative, this is equivalent to

$$\min_{\substack{x \in \{0,1\}^n \\ x_s = 1, x_t = 0}} \|\mathrm{diag}(w) B x\|_1 \tag{4.2}$$

where B is the incidence matrix of the graph.

From the sake of readability, on let $K := \mathrm{diag}(w)B$, i.e. K could be seen as some kind of 'weighted incidence matrix' of the graph.

The problem in (4.2) is still a binary problem and from the algorithmic point of view this makes it a hard problem. By dropping the binary constraint one gets a new (and hopefully easier) problem, the s.c. relaxed problem. In general the optimal solution of the binary problem and its relaxed version does not have to coincide. But for the graph cut problem this is the case, therefore, instead of solving (4.2) it is sufficient to solve

$$\min_{\substack{x\in[0,1]^n \\ x_s=1, x_t=0}} \|Kx\|_1 \tag{4.3}$$

In particular the solution of the continuous problem will be binary.

For graph cuts with directed graphs this results still holds, which is a basic result in linear optimization. This is due to the s.c. unimodularity of the (directed) max-flow/min-cut problem which is the criterion for integer solutions of linear programs, see for instance [13]. In section 4.4 a proof (not using the concepts from linear optimization) will be presented.

In the previous section where Lee's algorithm was presented the following formulation was used:

$$\min_{x\in\mathbb{R}^n} \|Kx\|_1 \qquad s.t. \quad \langle x, \chi\rangle = 1$$

where $\chi_s = 1, \chi_t = -1$ and $\chi_i = 0$ for all $i \neq s, t$, and therefore, the constraint simply states $x_s - x_t = 1$. In order to see that this formulation is equivalent to the previous ones, observe that for every optimal solution x^* the vector $x = x^* + 1\lambda$ is another solution, since $1 \in \ker(K)$. Therefore, it is justified to fix x_s and x_t:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \|Kx\|_1 \qquad s.t. \quad & x_s = 1 \\ & x_t = 0 \end{aligned}$$

The restrictions to the interval $[0, 1]$ is no real restriction, since one can easily check that for a vector $x \in \mathbb{R}^n$ the projection to $[0, 1]^n$ does not has a worse objective value.

4.1.2. Graph cuts in computer vision

In computer vision very often a different formulation is used. This is due to the fact that in computer vision images are often represented by a $M \times N$ regular grid graph and each pixel is represented by a vertex. Depending on the task, each pixel has a neighbourhood. In computer vision these neighbourhoods are 4-neighbourhoods or 8-neighbourhood

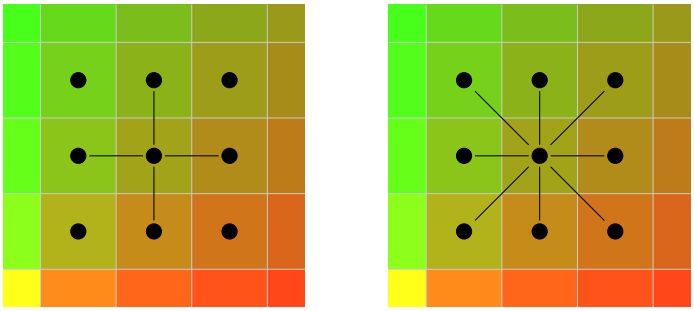


Figure 4.1.: The neighbours of a pixel in a 4 and an 8-neighbourhood

Finally very vertex is connected to the source s and to the sink t. See Figure 4.2 for an illustration of a typical computer vision cut.

For the case where 4-neighbourhoods of pixels are considered the incidence matrix of the vertices (without s and t) is given by the backward-derivative operator. With this observation the problem in (4.2) becomes:

$$\min_{x\in\{0,1\}} \left\|\operatorname{diag}(w^b)\nabla x\right\|_1 + \langle 1-x, w^s\rangle + \langle x-0, w^t\rangle$$

where ∇ is the gradient operator/backward difference, w^b are the weights between the pixels, w^s resp. w^t weights between the vertices and the source s resp. sink t. This can be further simplified to

$$\min_{x\in\{0,1\}} \left\|\operatorname{diag}(w^b)\nabla x\right\|_1 + \langle x, \underbrace{w^t - w^s}_{w^u}\rangle + \langle 1, w^s\rangle \tag{4.4}$$

The weights w^u are called unary weights and w^b are called binary weights, which makes sense if one restricts itself to the vertices corresponding to pixels and ignores the vertices s and t.

Furthermore, the last summand is a constant and can be dropped if one is only interested in the solution x and not its optimal value.

Note that by exchanging ∇ with the incidence matrix of the subgraph $V - \{s,t\}$ in (4.4) and setting w_i^s to zero if $(s,v_i) \notin E$ (and analogous for w^t) the problem in (4.4) is the same as in (4.2), hence every min-cut problem can be formulated with unary and binary weights.

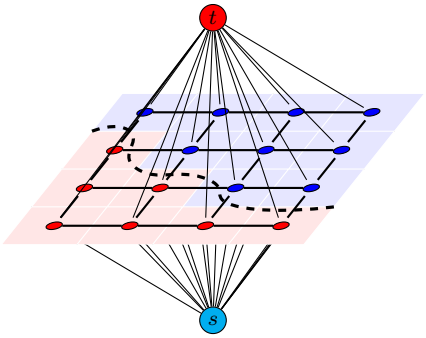


Figure 4.2.: An instance of a typical computer vision cut

4.2. Definitions

In this subsection some basic definitions and properties of convex functions are revised. All the algorithm presented later will depend on these properties. Since convex analysis and convex optimization are extensive theories in mathematics only the important definitions, required for the later presented algorithms, are discussed. For a comprehensive introduction into the whole theory see [51]. The fundamental definitions of convex sets, convex function and Lipschitz continuity was already given in section 1.1.

In the following let $X \subseteq \mathbb{R}^n$ denote a subset, for some $n \in \mathbb{N}$.

4.2.1. Further properties of functions

Only functions with Lipschitz continuous gradient will be studied.

Definition 30. (lower semi-continous)
A function $f : X \to \mathbb{R}$ is called lower semi-continuous, abbreviated lsc, iff

$$\liminf_{y \to x} f(y) \geq f(x)$$

Definition 31. (proper)
A function is called proper the function value is finite at least for one element, i.e.

$$\exists x \in X : f(x) < \infty$$

Definition 32. (spectral norm)
Let $A \in \mathbb{R}^{m \times n}$ then the spectral norm of A is defined as

$$\|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2 = \sqrt{\lambda_{\max}(A^t A)}$$

where $\lambda_{\max}$ denotes the largest eigenvalue. Furthermore, the spectral norm is equal to the largest singular value of A.

4.2.2. Subderivative, subgradients and subdifferentials

Recall that not every continuous function is differentiable. A well known candidate for such a function is the absolute value function, which is differentiable everywhere except at the origin.

Subderivative, subgradients and subdifferentials try to generalize the main properties from differential calculus for non differentiable and even non continuous functions.

First of all only convex functions will be considered in the next section. The derivative of a convex function, $f : \mathbb{R} \to \mathbb{R}$, at a certain point, describes the slope of a tangent line at that point. If f is differentiable, this tangent line is unique and given by $f(x) + \langle f'(x), y - x \rangle$. If f is not differentiable this line is not unique.

It is a well know fact, that a convex function has to lie over all its tangent lines, i.e. for a differentiable function $f : X \subseteq \mathbb{R} \to \mathbb{R}$ it holds

$$f(y) \geq f(x) + \langle f'(x), y - x \rangle$$

This property is used to define subderivatives:

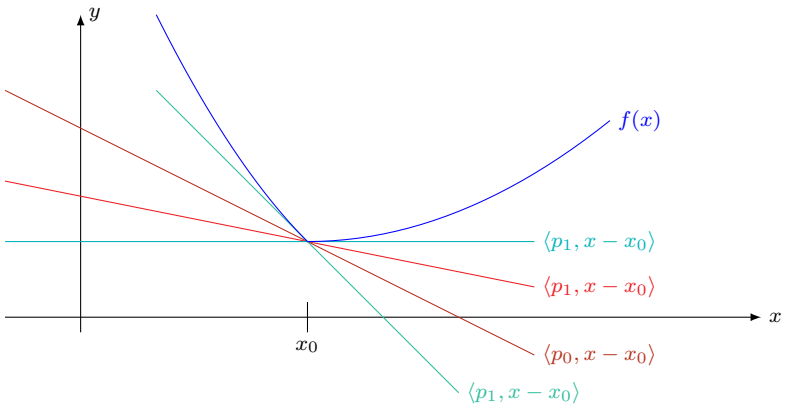


Figure 4.3.: two subderivatives at x

Definition 33. (Subderivative, -gradient, -differential)
For a open interval $X \subseteq \mathbb{R}$ and a function $f : X \to \mathbb{R}$, a *subderivative* of a function f at point $x \in X$ is a point $p \in \mathbb{R}$ such that

$$f(y) \geq f(x) + \langle p, y - x \rangle \qquad \forall y \in X$$

The *subgradient* is the generalization to higher dimensions, i.e. $X \subseteq \mathbb{R}^n$ and $p \in \mathbb{R}^n$. The *subdifferential* at a point x is the collection of all subgradients at this point and is denoted as $\partial f(x)$.

Note that if f is differentiable at x, then the subdifferential only consists of $\nabla f(x)$.

The nice property of the gradients and convex functions is that they characterize the minima, i.e. x^* is a minima iff $\nabla f(x^*) = 0$. Subdifferentials expand this property to non-differentiable functions.

Corollary 2. *Let f and X as in the previous definition, then*

$$x^* = \min_{x \in X} f(x) \iff 0 \in \partial f(x^*)$$

The proof follows immediately from the definition of the subdifferential. It is possible to use the subgradients directly in order to optimize the function f in gradient descent like fashion, i.e. update step is: $x^{i+1} = x^i + \alpha p$ with $p \in \partial f(x^i)$. However, this so called *subgradient method* has a very slow convergence rates [63].

4.2.3. Convex conjugate

The second important concept is the concept of the *convex conjugate* or also known as Fenchel transformation. Most of the following definitions will work on Banach spaces. But only the space $(\mathbb{R}^n, \|.\|_2)$ (which of course is a Banach space) will be considered, since this will be enough for our purposes. For the more general cases see again [12].

Notation 5. In order to shorten the notation let $\mathbb{R}_\infty := \mathbb{R} \cup \{\infty\}$.

The convex conjugate of a function is defined as follow:

Definition 34. (convex conjugate)
For a function $f : \mathbb{R}^n \to \mathbb{R}_\infty$ the convex conjugate, $f^* : \mathbb{R}^n \to \mathbb{R}_\infty$, is defined as

$$f^*(y) := \sup_{x \in \mathbb{R}^n} \langle x, y \rangle - f(x)$$

The convex conjugate is a convex function, since it is the piecewise supremum over convex functions.

A nice interpretation is given by the biconjugate function f^{**}, the conjugate of f^*. From the definition of the convex conjugate it immediately follows that

$$\begin{aligned} f^*(y) \geq \langle x, y \rangle - f(x) \Leftrightarrow f(x) &\geq \langle x, y \rangle - f^*(y) \\ &\overset{\sup_y}{\Longrightarrow} f(x) \geq f^{**}(x) \end{aligned}$$

indicating that the biconjugate is a convex function below the (possible non convex) function f. Moreover, it can be shown that the biconjugate is the largest convex function below f. If f was already convex $f^{**} = f$ follows.

4.3. Primal and dual problem and the primal-dual gap

4.3.1. Primal-dual problem

In this section the following minimization problem is considered:

$$\min_{x \in X} F(Kx) + G(x) \tag{P}$$

where $F : Y \subseteq \mathbb{R}^m \to \mathbb{R}_\infty$, $G : X \subseteq \mathbb{R}^n \to \mathbb{R}_\infty$ are proper, convex and lower semi-continuous functions. $K : X \to Y$ is a linear and continuous map, i.e. in this case a $(m \times n)$-matrix over $\mathbb{R}$. This problem is called the *primal problem*.

If one assumes that the primal problem has a solution, the minimum can be replaced with the infimum. Furthermore, since F is convex $F = F^{**}$ holds and the primal problem can be rewritten as

$$\inf_{x \in X} \sup_{y \in Y} \langle x, K^t y \rangle - F^*(y) + G(x) \tag{4.4}$$

if one assumes that everything is nice (i.e.. the supremum and infimum can be exchanged and a maximum exists) it follows:

$$\begin{aligned}
&\max_{y\in Y} -F^*(y) + \inf_{x\in X}\langle x, K^t y\rangle + G(x)\\
&\max_{y\in Y} -F^*(y) - \sup_{x\in X}\langle x, -K^t y\rangle - G(x)\\
&\max_{y\in Y} -F^*(y) - G^*(-K^t y)
\end{aligned}$$

The problem in the last line is called the *dual problem*. The calculations reveal that the dual problem can be seen as rewriting Problem (P) in terms of the convex conjugates of F and G. Furthermore, the terms primal and dual could be interchanged easily, but in general, the primal problem is the given problem to be solved (hence the name).

Of course the above calculation is simplified and will not hold in general. Nevertheless, the following theorem specifies the conditions and states that the primal problem and the dual problem are equivalent.

Theorem 15 (Fenchel Rockafellar duality)**.** *Let $F : Y \subseteq \mathbb{R}^m \to \mathbb{R}_\infty$ and $G : X \subseteq \mathbb{R}^n \to \mathbb{R}_\infty$ proper, convex and lower semi-continuous. Furthermore, let $K \in \mathbb{R}^{m\times n}$ and x^* a solution of the primal problem:*

$$\min_{x\in\mathbb{R}} F(Kx) + G(x)$$

If there exists a point $\bar{x} \in X$ such that $F(K\bar{x}), G(\bar{x}) < \infty$ and F continuous at $K\bar{x}$ then the primal problem and the dual problem are equal, i.e.

$$\min_{x\in\mathbb{R}} F(Kx) + G(x) = \max_{y\in Y} -F^*(y) - G^*(-K^t y)$$

Furthermore, the pair of the optimal solutions can easily be characterised, this characterisation is given by the next corollary.

Corollary 3. *With the same requirements as in Theorem 15, let $x^* \in X$ and $y^* \in Y$. The pair (x^*, y^*) is a solution of the primal and dual problem iff*

$$-K^t y^* \in \partial G(x^*) \quad \wedge \quad y^* \in \partial F(Kx^*)$$

For a proof of Theorem 15 and Corollary 3 see [12] or (with a quite different notation [51]).

4.3.2. The minmax Problem and Saddle Points

The pair of solutions (x^*, y^*) is often called a *saddle point*. From the previous definitions and calculations it is not quite clear why this is the case.

The standard definition of a saddle point says that x is a saddle point of a function f, if the first derivative and the second derivative are zero. Or in higher dimensions the gradient has to be zero and the Hesse matrix indefinite. Recall that the functions F and G are only lower semi-continuous and therefore not necessarily differentiable, for this case there exists a different definition.

Definition 35. (Saddle point)
Let $f : X \times Y \to \mathbb{R}_\infty$, where $X \subseteq \mathbb{R}^n$, $Y \subseteq \mathbb{R}^m$. A pair $(\bar{x}, \bar{y}) \in X \times Y$ is called a saddle point of f if

$$f(\bar{x}, y) \leq f(\bar{x}, \bar{y}) \leq f(x, \bar{y}) \qquad \forall x \in \bar{X} \quad \forall y \in \bar{Y}$$

or if

$$f(x, \bar{y}) \leq f(\bar{x}, \bar{y}) \leq f(\bar{x}, y) \qquad \forall x \in \bar{X} \quad \forall y \in \bar{Y}$$

where $\bar{X}, \bar{Y}$ are some (topological) neighbourhood of X and Y.

The two different conditions of a saddle point arise from the fact that one could exchange the order of the coordinates, i.e. $g(y, x) := f(x, y)$ and $(\bar{x}, \bar{y})$ would still be a saddle point but the inequalities for f would not hold for g any more, see Figure 4.4.

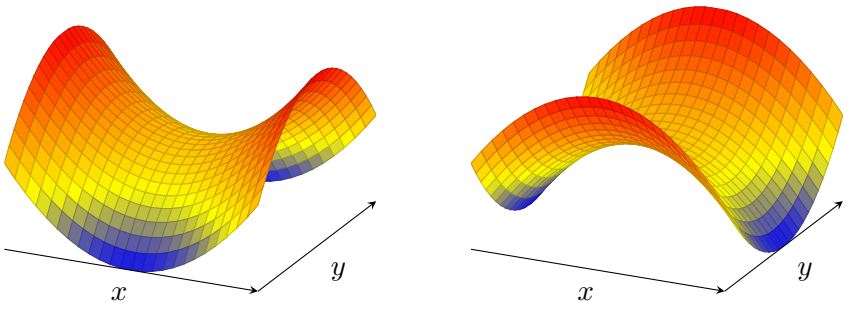


Figure 4.4.: Saddle point of $f(x, y) = x^2 - y^2$ and $f(x, y) = y^2 - x^2$

To see that Definition 35 indeed generalizes the definition of saddle points in the differentiable case, consider a function $f : \mathbb{R}^2 \to \mathbb{R}_\infty$ with (x^*, y^*) as a saddle point. By definition it holds that

$$f(x^*, y) \leq f(x^*, y^*) \leq f(x, y^*) \tag{4.5}$$

for x, y in a neighbourhood of (x^*, y^*). This indicates that x^* (resp. y^*) is a local minima (resp. maxima) of $f|_{y=y^*}(x, y)$ (resp. $f_{x=x^*}(x, y)$). Therefore, the gradient of f is zero, but since $f(x^*, y^*)$ is not a minima nor a maxima the Hesse matrix has to be indefinite. This argument can of course be generalized to higher dimensions, i.e. $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$.

This motivates the following corollary

Corollary 4. *Let*

$$L : X \times Y \to \mathbb{R}_\infty \qquad L(x, y) := \langle Kx, y \rangle - F^*(y) + G(x)$$

then the pair (x^*, y^*) is a solution of the primal and dual problem iff it is a saddle point of L.

Proof. Since it is assumed that a solution of the primal and dual problem exists, they are equivalent to the problem given in (4.3.1) which is $\inf_{x\in X}\sup_{y\in Y} L(x,y)$. Now it follows that

$$L(x^*,y) \leq \sup_{y\in Y} L(x^*,y) = L(x^*,y^*) = \inf_{x\in X} L(x,y^*) \leq L(x,y^*)$$

On the other hand, if (x^*,y^*) is a saddle point it follows from (4.5) that

$$L(x^*,y^*) \geq L(x^*,y) \Rightarrow L(x^*,y^*) \geq \sup_{y\in Y} L(x^*,y)$$
$$L(x^*,y^*) \leq L(x,y^*) \Rightarrow L(x^*,y^*) \leq \inf_{x\in X} L(x,y^*)$$

hence

$$\sup_{y\in Y} L(x^*,y) \leq \inf_{x\in X} L(x,y^*)$$

Applying the Fenchel transformation

$$F(Kx^*) + G(x^*) \leq -F^*(y^*) - G^*(-K^t y^*)$$

follows.

Theorem 15 indicates that the left-hand side is always greater than the right-hand side, therefore equality follows and the the saddle point really is a optimal solution. □

Combining this corollary with the definition of a saddle point one gets another equivalent formulation of the primal problem. The problem of finding a saddle point is simply called *saddle point problem* and is given by the following minmax problem

$$\min_{x\in X}\max_{y\in Y} \langle Ax,y\rangle - F^*(y) + G(x) \qquad \text{(MinMax)}$$

4.3.3. Primal and dual energy, primal-dual gap

The next definition is about the *energy* of a function, in terms of optimization this is the same as the value of the objective function. But to coincide with the literature the term energy will be used in this thesis.

Definition 36. (primal energy, dual energy)
For a given $x \in X$ the *primal energy*, denoted by $\mathfrak{p}(x)$, is defined as

$$\mathfrak{p}(x) := F(Kx) + G(x)$$

analog for $y \in Y$, the *dual energy*, denoted by $\mathfrak{d}(y)$, is defined as

$$\mathfrak{d}(y) := -F^*(y) - G^*(-K^t y)$$

From Theorem 3 it is clear that

$$\mathfrak{p}(x) \leq \mathfrak{d}(y) \qquad \forall x \in X \quad y \in Y$$

This motivates the definition of the so called *primal-dual gap* or sometimes also called *duality gap*.

Definition 37. (primal-dual gap)
Let $(x, y) \in X \times Y$ then the value of $\mathfrak{p}(x) - \mathfrak{d}(y)$ is called the primal-dual gap at (x, y) and is denoted by $\mathcal{G}(x, y)$

With the same argument used to show the connection between the primal and the dual problem, a different definition of the primal/dual energy could be given, by writing F, G in terms of their convex conjugate. If $F(Kx)$ is rewritten in terms of the convex conjugate one gets

$$\mathfrak{p}(x) = \sup_{y \in Y} \langle Kx, y \rangle - F^*(y) + G(x)$$

similar for the dual energy and the convex conjugate of $G^*(-K^t y)$

$$\mathfrak{d}(y) = \inf_{x \in X} \langle x, K^t y \rangle - F^*(y) + G(x) \tag{4.6}$$

Now the saddle point problem (MinMax) is equal to minimizing resp. maximizing the primal resp. dual energy.

Sometimes it is easier to consider the primal-dual gap only on a subset of X and Y, this leads to the definition of the *restricted primal-dual gap*:

Definition 38. (restricted primal-dual gap)
For two sets $B_1 \subseteq X$, $B_2 \subseteq Y$ the *restricted* primal-dual gap is defined as

$$\mathcal{G}_{B_1, B_2}(x, y) = \sup_{\bar{y} \in B_2} \langle Kx, \bar{y} \rangle - F^*(\bar{y}) + G(x) - \inf_{\bar{x} \in B_1} \langle \bar{x}, K^t y \rangle - F^*(y) + G(\bar{x})$$

It is clear that the restricted gap is always larger or equal to the (unrestricted) gap, as soon as $B_1 \times B_2$ contains a saddle point.

4.3.4. Proximal map

The proximal map can be seen as a generalization of a projection onto a (convex) set.

Definition 39. (proximal operator, proximal map)
Let $f : X \to \mathbb{R}$ be a convex function, then the *proximal operator* or *proximal map* is given by

$$\operatorname{prox}_f(x) = \arg\min_{y \in X} \frac{\|x - y\|_2^2}{2} + f(y) \tag{4.7}$$

If f is the characteristic function of a convex set, the proximal maps becomes a projection onto this set. Notice that the minimization problem in the definition is strictly convex, therefore, $\text{prox}_f(x)$ is unique. In the general case the proximal map of a point x is a point with a better function value but still 'close' to x.

Using subgradient calculus one can easily derive the optimality condition for the right-hand side of (4.7)

$$\begin{aligned} 0 &\in -x + y + \partial f(y) \\ x &\in y + \partial f(y) \\ x &\in (I + \partial f)(y) \\ y &= (I + \partial f)^{-1}(x) \end{aligned}$$

this allows to write $\text{prox}_f(x) = (I + \partial f)^{-1}(x)$.

One identity which will be useful for calculating proximal maps of convex conjugated functions, is the Moreau's identity given by:

Corollary 5 (Moreau's identity)**.** *For $\sigma > 0$ it holds that*

$$x = \text{prox}_{\sigma f}(x) + \sigma \, \text{prox}_{1/\sigma f^*}(x/\sigma)$$

A proof for $\sigma = 1$ is given in [51], the case for general σ is obtained by substituting σf for f.

4.4. Equivalence between the relaxed and the binary problem

In this section it is shown that the binary problem and the continuous relaxed problem have the same solution, in particular the solution of the continuous problem is binary. The proof of this properties, is from a paper of Chambolle [14] and it works not only for graph cuts but also for the more wider class of minimization problems of the form:

$$\min_{x \in \{0,1\}^n} \lambda J(x) + \langle s - G, x \rangle \tag{P_s}$$

where $J : \mathbb{R}^n \to [0, \infty]$ is a convev, lsc, positively one-homogenious function satisfying the s.c. *generalized coarea formula*. A positively k-homogenious function simply satisfies $J(\alpha u) = \alpha^k J(u)$ for $\alpha > 0$. The generalized coarea formula is a little bit more sophisticated and defined as

$$J(u) = \int_{-\infty}^{\infty} J([u > t])dt$$

where the Iverson brackets $[u > t]$ are understand pointwise, i.e. $[u > t] := \big([u_i > t]\big)_{i=1}^n$ and $[u_i > t]$ is one if $u_i > t$ and zero otherwise.

The following proposition states that the optimal solution of a (binary) problem with the same form as (P_s) is also a solution of the relaxed problem.

Proposition 1 (Chambolle[14]). *Any solution of* (P_s) *is a solution of*

$$\min_{x\in[0,1]^n} \lambda J(x) + \langle s - G, x\rangle \tag{$\bar{P}_s$}$$

and vice versa.

Proof. Note that for $c \in \mathbb{R}$ and the vector $y = (c, c, \cdots, c)$, $J(y) = 0$, otherwise $J(x) = -\infty$ for all $x \in R^n$, which would be a boring minimization problem. Let $x \in [0,1]^n$, it follows that

$$\begin{aligned}\lambda J(x) &= \int_{-\infty}^{\infty} \lambda J([x > t])dt \\ &= \int_0^1 \lambda J([x > t])dt + \int_{-\infty}^0 \lambda J(0)dt + \int_1^{\infty} \lambda J(1)dt \\ &= \int_0^1 \lambda J([x > t])dt\end{aligned}$$

Furthermore, $x_i = \int_0^1 [x_i > t]dt$, this allows to rewrite the inner product:

$$\langle s - G, x\rangle = \sum_{i=1}^n (s - G_i)x_i = \sum_{i=1}^n (s - G_i)\int_0^1 [x_i > t]dt = \int_0^1 \langle s - G, [x > t]\rangle dt$$

Putting these two conversions together, it follows:

$$\lambda J(x) - \langle s - G, x\rangle = \int_0^1 \Big\{\lambda J([x > t]) - \langle s - G, [x > t]\rangle\Big\}dt \tag{4.8}$$

For the sake of readability lets rewrite equation (4.8) as $f(x) = \int_0^1 f([x > t])dt$.

First, let x^* be a solution of the binary problem (P_s), clearly it follows that $f(x^*) \leq f([x > t])$ for all $t \in [0,1]$ and for all $x \in \mathbb{R}^n$. Therefore minimizing both sides for x yields:

$$\min_{x\in[0,1]^n} f(x) = \min_{x\in[0,1]^n} \int_0^1 f([x > t])dt \geq \int_0^1 f(x^*)dt = f(x^*) := \min_{x\in\{0,1\}^n} f(x)$$

Equality follows since the left problem is a relaxation of the right one.

Now let x^* be a solution of $\min_{x\in[0,1]^n} f(x)$. Then of course:

$$f(x^*) = \min_{x\in[0,1]^n} f(x) \leq f([x^* \geq t]) \tag{4.9}$$

for all t. From (4.8) we know that $f(x^*) = \int_0^1 f([x^* > t])dt$ and and since $f(x^*) = \int_0^1 f(x^*)dt$, inequality (4.9) yields:

$$\int_0^1 \underbrace{f([x^* > t]) - f(x^*)}_{\geq 0} dt = 0$$

indicating $f([x^* > t]) = f(x^*)$ almost everywhere. As a function of t, $f([x^* > t])$ is almost everywhere constant. Furthermore, the function value only (but not necessarily) changes on the entries of x^*. Let $x^*_{\sigma(i)}$ denote the i-th smallest entry of x^*, then the function is constant on right-closed intervals $\left(x^*_{\sigma(i)}, x^*_{\sigma(i+1)}\right]$ (this hold even for $x^*_{\sigma(i+1)} = x^*_{\sigma(i)}$). Therefore,

$$\bigcup_{i=1}^{n-1} \left(x^*_{\sigma(i)}, x^*_{\sigma(i+1)}\right] \cup (-\infty, x^*_{\sigma(1)}] \cup (x^*_{\sigma(n)}, \infty) = \mathbb{R}$$

$f([x^* > t]) = f(x^*)$ holds for all $t \in \mathbb{R}$.

Together with (4.9) it follows that $f([x^* > t]) = \min_{x \in [0,1]^n} f(x)$. □

By simple calculations one can check that $\left\|\operatorname{diag} w^b B x\right\|_1$ satisfies the coarea formula and the rest of the requirements for J. Setting $s = 0$ and $G = -w^u$ reveals that the min-cut problem actually can be formulated in the form given in (P_s). Therefore, the restrictions to binary solutions can be dropped in (P_s).

5. The primal-dual algorithm and its derivates

This section covers the primal-dual algorithm of Pock and Chambolle [15] and the basic algorithm for optimizing the saddle point problem MinMax is given. Furthermore, the iteration steps for the graph cut problem will be derived, a method for speeding up the primal-dual algorithm for binary problems and how to calculate $(1+\varepsilon)$-approximations will be discussed.

5.1. The standard algorithm

The primal-dual algorithm is used to minimize problems of the form (P). The name 'primal-dual algorithms' originates from the fact that these algorithm optimize the primal and the dual problem in alternating fashion. Hence, a solution for the dual problem is calculated as well.

Recall that it was required, that $F : Y \subseteq \mathbb{R}^m \to \mathbb{R}_\infty$, $G : X \subseteq \mathbb{R}^n \to \mathbb{R}_\infty$ are proper, convex and lower semi-continuous functions and $K : X \to Y$ is a linear and continuous map.

Now, given with the definitions from the previous section the primal-dual algorithm, in its most basic form, can be written down in a very compact form:

Algorithm 8 Primal-dual algorithm

1: choose $\sigma, \tau > 0$
2: choose $(x^0, y^0) \in X \times Y$
3: $\bar{x}^0 = x^0$, $k = 0$
4: **while** not converged **do**
5: $\quad y^{k+1} = (I + \sigma \partial F^*)^{-1}(y^k + \sigma K \bar{x}^k)$
6: $\quad x^{k+1} = (I + \tau \partial G)^{-1}(x^k - \tau K^t y^k)$
7: $\quad \bar{x}^{k+1} = 2x^{k+1} - x^k$
8: $\quad k \leftarrow k + 1$

In line 5 and 6 the calculation of a proximal map is required. Hence, this calculations should be possible and not too time consuming. Note, that because of the Moreau's identity, one does not have to explicitly calculate the proximal map of F^*. Therefore, a simple proximal map of F, G may be stated as an additional requirement for the algorithm.

In line 6 the algorithm updates the current dual solution and in line 5 the primal solution is updated. With this observation the primal-dual gap can serve as a stopping criterion. If the gap is zero (or small enough) a saddle point is found.

The following theorem states the convergence rate of the algorithm

Theorem 16 (Primal-dual algorithm (Chambolle and Pock [15, 17]))**.** *Let $L = \|K\|$ and choose σ and τ such that $\sigma\tau L^2 < 1$. Furthermore, assume that Problem (MinMax) has a solution and denote this solution by (x^*, y^*). Let $n, x^n, y^n, \bar{x}^n$ as in Algorithm 8, then it holds that*

(i) for any $k > 0$

$$\frac{\left\|y^k - y^*\right\|_2^2}{2\sigma} + \frac{\left\|x^k - x^*\right\|_2^2}{2\tau} \leq (1 - \sigma\tau L^2)^{-1} \left(\frac{\|y^0 - y^*\|_2^2}{2\sigma} + \frac{\|x^0 - x^*\|_2^2}{2\tau} \right)$$

(ii) Let $\tilde{x}^N := \sum_{k=1}^N x^k/N$, analog $\tilde{y}^N := \sum_{k=1}^N y^k/N$ and $B_1 \times B_2 \subseteq X \times Y$ a bounded set. Then the restricted primal-dual gap is bounded by

$$\mathcal{G}_{B_1 \times B_2}(\tilde{x}^N, \tilde{y}^N) \leq \sup_{(x,y) \in B_1 \times B_2} \frac{1}{N} \left(\frac{\|y^0 - y\|_2^2}{2\sigma} + \frac{\|x^0 - x\|_2^2}{2\tau} - \langle K(x - x^0), y - y^0 \rangle \right)$$

(iii) if the dimension of X and Y is finite the $(x^k)_{k \geq 0}$ and $(y^k)_{k \geq 0}$ converge to a saddle point.

Despite its simple structure the proof of convergence is quite long and the algorithm is only guarantied to converge on average. Recall that, if $B_1 \times B_2$ contains a saddle point, the primal-dual gap is always smaller than the restricted primal-dual gap, hence (ii) gives a real upper bound of the gap. Together with (iii) it follows that, in finite dimensions, the algorithm converges with rate $O(1/N)$. Later, when discussing how to calculate approximate solutions of min-cut problems, (ii) will prove useful again in order to estimate the quality of approximation.

5.2. Primal-dual graph cut formulations and updates

In this section it is discuss how to solve graph cut problems with the primal-dual algorithm, i.e. how graph cuts can be written in the form of problem (P).

5.2.1. The standard Problem formulation

All the previous problem formulations were constraint problems. Moreover, the constraints were of the form $x \in A$ for some convex set A. Problems with such constraints can easily be converted into unconstraint problems by using the characteristic function.

For a set $A \subseteq \mathbb{R}^n$ the characteristic function χ_A is defined as

$$\chi_A(x) := \begin{cases} 0 & x \in A \\ \infty & x \notin A \end{cases}$$

This functions allows the elegant formulation of the problem given in (4.3) as

$$\min_{x \in \mathbb{R}^n} \underbrace{\|\mathrm{diag}(w)Bx\|_1}_{:=F(Kx)} + \underbrace{\chi_C(x)}_{:=G(x)}$$

where $C = \{c \in [0,1]^n | c_s = 1 \ , \ c_t = 0\}$. While the choice of G is clear, for F one can ether choose $F := \|\cdot\|_1$ or $F := \|\mathrm{diag}(w)\cdot\|_1$. The update steps for the two different cases are slightly different. For the case $F = \|\cdot\|_1$ they are given by:

$$\begin{cases} y^{k+1} = \mathcal{P}_{[-1,1]}(y^k + \sigma K(2x^k - x^{k-1})) \\ x^{k+1} = \mathcal{P}_C(x^k - \tau K^t y^{k+1}) \end{cases}$$

for the other case:

$$\begin{cases} y^{k+1} = \mathcal{P}_{[-w,w]}(y^k + \sigma B(2x^k - x^{k-1})) \\ x^{k+1} = \mathcal{P}_C(x^k - \tau B^t y^{k+1}) \end{cases}$$

Not required by the algorithm itself but useful for the stopping criterion is the dual energy. From the definition of the dual energy, given in (4.6), it follows that:

$$\begin{aligned} \mathfrak{d}(y) &= \inf_{x \in \mathbb{R}^n} \langle x, K^t y \rangle - F^*(y) + G(x) \\ &= \inf_{x \in \mathbb{R}^n} \langle x, K^t y \rangle - \chi_{[-1,1]^m}(y) + \chi_C(x) \end{aligned}$$

since $y \in [-1,1]^m$ during the algorithm, we can further simplify

$$\mathfrak{d}(y) = \inf_{x \in C} \langle x, K^t y \rangle = \inf_{x \in C} \sum_{i=1}^{n} x_i (K^t y)_i$$

The infimum is attained if $x_i = 0$ if $(K^t y)_i > 0$ and $x_i = 1$ otherwise, for all $i \notin \{s,t\}$. Note that the formula for the dual energy is the same for both cases $F = \|\cdot\|_1$ and $F = \|\mathrm{diag}(w)\cdot\|_1$ (but of course K is different).

Now one can try to give an interpretation of the dual variable or the dual energy. Because if formulated as linear program the dual of the max-flow problem is the min-cut problem (and vice versa). But in this case one does not has a linear program anymore and the duality is not the same as in linear programs. The dual problem is to maximize the dual energy and therefore

$$\max_{y \in [-1,1]} \sum_{\substack{i=1 \\ i \neq s,t}}^{n} (K^t y)_i [(K^t y)_i \leq 0] + (K^t y)_s$$

Now observe that in general the term $(K^t y)_i$ should be maximized but only until they they are positive. Furthermore, if one thinks of y as something like a flow, then $(K^t y)_i$ is the overflow at vertex i, i.e. the sum of the inflow minus the outflow. A value less than zero means that more 'flow' is leaving the vertex than entering it, and as soon as the overflow gets positive interest for this vertex is lost. Or in other words, try to achieve a nonnegative overflow at all vertices.

5.2.2. Computer vision formulations

The update steps for computer vision graphs are only slightly different

$$\begin{cases} y^{k+1} = \mathcal{P}_{[-w^b, w^b]^n}(y^k + \sigma\nabla(2x^k - x^{k-1})) \\ x^{k+1} = \mathcal{P}_{[0,1]^n}(x^k - \tau(\nabla^t y^{k+1} + w^u)) \end{cases} \tag{5.1}$$

Note that in this case $F := \left\|\mathrm{diag}(w^b)\cdot\right\|_1$ was used.

The dual energy for the computer vision graph formulation is given as:

$$\mathfrak{d}(y) = \inf_{x\in[0,1]^n} \langle x, w^u + K^t y\rangle = \sum_{i=1}^{n} (w^u + K^t y)_i [(w^u + K^t y)_i < 0]$$

where the calculations are the same as in the standard case.

5.2.3. Stopping criterion

In most implementation the primal-dual algorithm is stopped if the primal-dual gap is smaller than a certain tolerance. First of all this is because often a approximation of the problem is enough, secondly due to hardware restrictions the calculations cannot be performed exactly and small rounding errors are imminent. Because of the hardware restrictions it is even possible that the primal-dual gap never becomes exactly 0. For the graph cut problems binary solutions are needed, therefore, the result is rounded to 0 or 1. The question now is of course when to stop the calculations with respect to the primal-dual gap.

Assume that the weights are rounded to an increment of γ, i.e. the weights are chosen from the set $\{\gamma z | z \in \mathbb{Z}\}$. Let x^N be the solution after N iterations and let $\bar{x}^N$ be the vector resulting from optimal rounding x^N to binary values (the 'current binary solution'). Now, if $\mathcal{G}(\bar{x}^N, y^N) < \gamma$ then $\bar{x}^N$ is optimal since the smallest possible change in the cut values is at least γ. Note that it is important to use a binary solution but obtaining the optimal rounding strategy could be challenging because simply rounding the values to the next integer (i.e. thresholding at 0.5) is not necessarily the best strategy. See figure 5.1 for an illustration. Of course one can always obtain a binary solution by thresholding not only at 0.5 but any other value between 0 and 1 and iterate until the gap for this solution is less than γ.

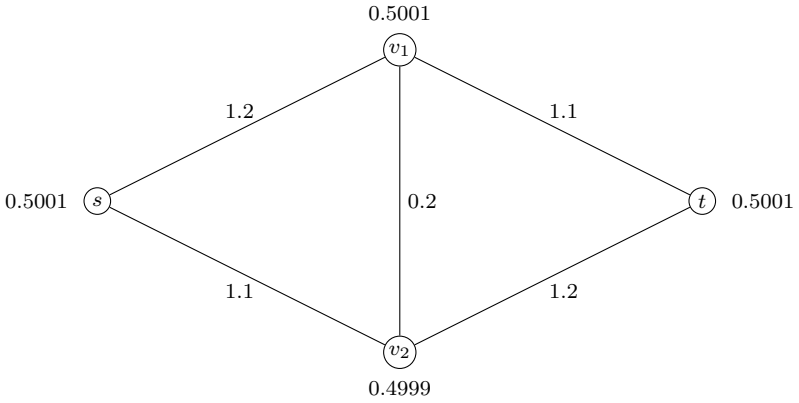


Figure 5.1.: Example showing how the value of a cut can change by an arbitrary small number, the optimal cut value is 2.3 and the current cut value is 2.30004 but simple thresholding at 0.5 does yield a cut with value 2.4, a better threshold would be at a value > 0.5001 (or < 0.4999)

Sweep Cuts

Random thresholding is not optimal but is in a way justified by the following theorem from the paper of Christiano et al. [19]:

Theorem 17. *Given a vector $\phi \in [0,1]^n$, where $\phi_s = 1$ and $\phi_t = 0$. Then the expected value of the cut, obtained by thresholding ϕ at a value randomly chosen from the interval $[0,1]$, is exactly*

$$\sum_{(u,v)\in E} |\phi_u - \phi_v| \, w(u,v) \tag{5.2}$$

Furthermore this implies that there exists a $t \in [0,1]$ such that the value of the cut obtained by thresholding at t is better or equal to the expected value.

Proof. Since x is chosen uniformly and the probability that a edge $(u,v) \in E$ is cut is exactly $|\phi_u - \phi_v|$. Therefore, the expected value of the cut is given by (5.2) and indicates that there exists a cut whose value is less than or equal to (5.2). □

If one is rather interested in the optimal thresholding value one can calculate the cut for every of the n thresholding values and pick the best. This can be achieved in an iterative fashion by first sorting the potentials and afterwards iteratively adding vertices to S and updating the current cut value. The change in each iteration can be calculated by examining all edges, incident with the currently added vertex and add/subtract the weight from the cut value. All these steps are summarized in Algorithm 9.

If the graph is very sparse the runtime is dominated by sorting the potential vector (in $O(n \log n)$ time), otherwise each edge is checked twice (in $O(m)$ time), therefore, the total runtime is $O(\max\{m, n \log n\})$.

Algorithm 9 SweepCut

Input: potential vector $x \in \mathbb{R}^n$
Output: minimal cut that can be obtained by thresholding x

```
x ← Sort(x), c* ← ∞, c ← 0                          ▷ x_0 ≈ s
for v = 0, ..., n do
    for u ∈ δ(v) do
        if v > u then
            c ← c + w((u, v))
        else
            c ← c − w((u, v))
    if c < c* then
        c* ← c
return {i | x_i > c*}
```

5.3. Global Relabeling for Graph Cuts

Unger et al. [62] presented a possible way to speed up the primal-dual algorithm for problems which have a binary solution. The main idea is to threshold the current continuous solution in order to get a binary solution. If this binary solution is significantly better, the algorithm continues with the obtained binary solution, otherwise the binary solution is discarded. Unger et al. called this process 'global relabeling'.

This strategy is motivated by the observation that the convergence of primal-dual gap is very fast at the start but becomes slow towards the end. In figure 5.2 the evolution of the primal-dual gap during the algorithm is plotted and it can be clearly seen that convergence of the standard primal-dual algorithm is taking quite long. In contrast the global relabeling significantly speeds up the convergence and after less than 500 iterations the optimal value is found.

Of course as illustrated in figure 5.1 finding the optimal thresholding value can be challenging. Therefore, Unger et al. proposed to threshold only at the values $\{0, 0.01, 0.05, 0.1, 0.5, 0.9, 0.95, 0.99\}$, which already yields good results. Relabeling at every iteration does not makes sense since the runtime would increase drastically. Therefore in their paper Unger et al. suggested to perform a relabeling step every $I = \max(M, N)$ iterations, where M is the image heights and N is the image widths.

Since the quality of the current solutions is measured with the respect to the normalized primal-dual gap and after each thresholding the primal variable changes severely, the dual variable has to be updated as well. Updating the dual variable is the same as in the primal-dual algorithm with the exception that a higher value of σ is used.

Furthermore, a parameter $\mu \in [0, 1]$ is used in order to perform a relabeling step only if the new binary solution would be a significant improvement.

Algorithm 10 shows the global relabeling algorithms, $\bar{y}_t^k$ denotes the updated dual variable for the binary solution $[x^k > t]$.

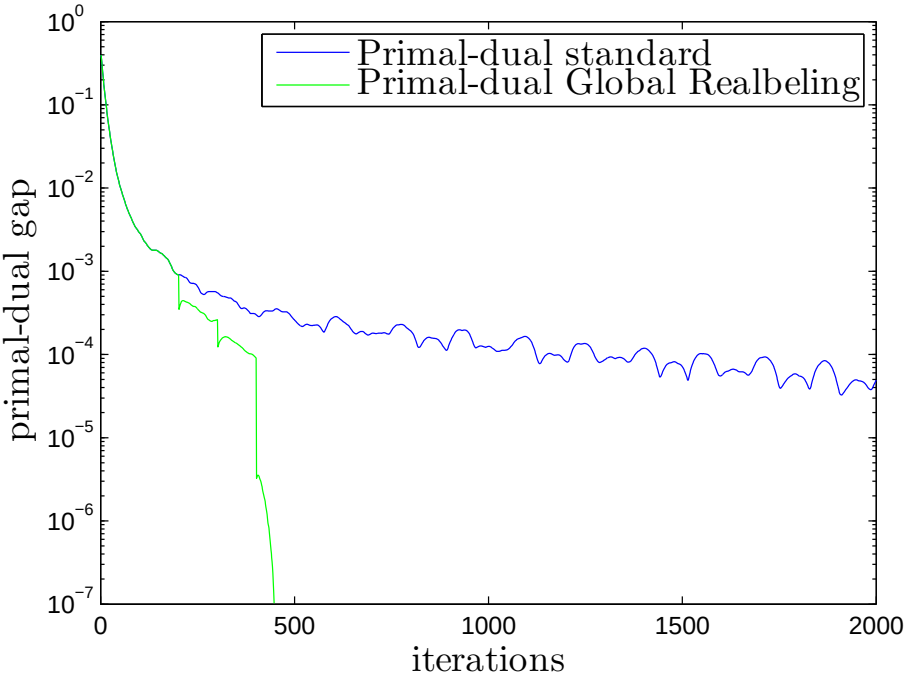


Figure 5.2.: Evolution of the primal-dual gap during the algorithm, after every 100 iterations a global relabeling step is performed

5.4. Calculating $(1+\varepsilon)$-Approximations of Min-Cut Problems

If one is interested in the rate of convergence of the primal-dual Algorithm, the second statement of Theorem 16 gives some insights about the partial primal-dual gap. After N iterations it holds:

$$\mathcal{G}_{B_1,B_2}(x_N, y_N) \leq \frac{1}{N}\left(\sup_{(x,y)\in B_1\times B_2} \frac{\|x-x^0\|^2}{2\tau} + \frac{\|y-y^0\|^2}{2\sigma}\right)$$

Algorithm 10 Global relabeling

```
choose I, choose μ
while not converged do
    for I iterations do
        do primal-dual update steps for x^k, y^k
    G_min ← min{G_min, G(x^k, y^k)}
    (x̃, ỹ) ← arg min_{t∈(0,1)} G(x^k > t, ȳ_t^k)
    if G(x̃, ỹ) < μ G_min then
        (x^k, y^k) ← (x̃, ỹ)
```

For $x_N = \sum_{k=1}^{N} x_k/N$, $y_N = \sum_{k=1}^{N} y_k/N$ and $B_1 \times B_2 \subset X \times Y$.

Since the graph cut problem is bounded (observe that both update steps involve projections to bounded sets) one can simply replace B_1 resp. B_2 by $[0,1]^{|V|}$ resp. $[-1,1]^{|E|}$ (or $[-w^b, w^b]$).

Since the graph cut problem has a solution a saddle point is containd in these sets, therefore, the primal-dual gap is smaller then the partial primal-dual gap.

Let again F^* denote the optimal value of the optimal (combinatorial) flow/cut, and (x^N, y^N) the solutions after N iterations. Same as in the previous sections $\mathfrak{p}(x^N)$ denotes the primal energy (the value of the cut) and $\mathfrak{d}(y^N)$ denotes the dual energy. The current solution x^N is a $(1+\varepsilon)$- approximation if $\mathfrak{p}(x^N) \leq (1+\varepsilon)F^*$. The same holds for the dual variable: if $\mathfrak{d}(y^N) \geq (1-\varepsilon)F^*$ then y^N is a $(1-\varepsilon)$-approximation.

Now for x^N and y^N two $(1 \pm \varepsilon)$-approximations the primal-dual gap is bounded by:

$$\mathcal{G} = \mathfrak{p}(x^N) - \mathfrak{d}(y^N) \leq (1+\varepsilon)F^* - (1-\varepsilon)F^* = 2\varepsilon F$$

However, of practical interest is the other direction. Since the primal and dual variable may converge with different speed this inequality would only grant that one variable is a $(1 \pm \varepsilon)$-approximation. Suppose that the primal variable is optimal, i.e. $\mathfrak{p}(x^N) = F^*$ then error of $\mathfrak{d}(y^N)$ could still be $2\varepsilon F^*$ without violating the inequality. See Figure 5.3 for a illustration.

In order to ensure that the primal energy and the dual energy is a $(1 \pm \varepsilon)$- approximation one has to ensure that:

$$\mathcal{G}(x^N, y^N) \leq \varepsilon F^*$$

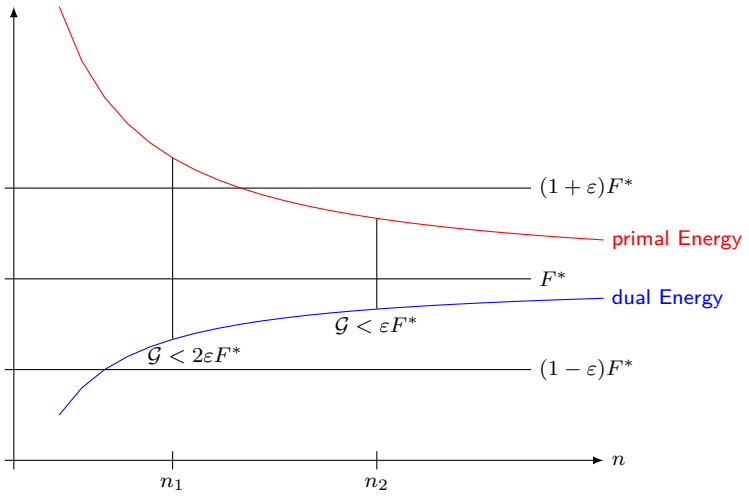


Figure 5.3.: An example evolution of the primal and the dual energy during the primal-dual algorithm; after n_1 iterations the dual variable is already a $(1-\varepsilon)$-approximation but the primal is no $(1+\varepsilon)$-approximation yet

Combining this observation with Theorem 16 it follows that algorithm has to be run until:

$$\frac{1}{N} \sup_{x,y \in B_1 \times B_2} \frac{\|x - x^0\|^2}{2\tau} + \frac{\|y - y^0\|^2}{2\sigma} \leq \varepsilon F^*$$

or

$$N \geq \frac{1}{\varepsilon F^*} \sup_{x,y \in B_1 \times B_2} \frac{\|x - x^0\|^2}{2\tau} + \frac{\|y - y^0\|^2}{2\sigma}$$

Recall that B_2 was ether $[-1,1]^m$ or $[-w^b, w^b]$. In the first case the supremum is $O(m)$ indicating that the number of iterations could be bounded by is $O(m/\varepsilon F^*)$, which is quite high. In the second case the supremum is $O(\|w^b\|) = O(m \max(w_i^b)^2)$. Now one could try to scale down w_b in order to get a better bound at the iterations as $O(m)$, i.e. scale w^b such that $\max_i(w_i^b)^2 = O(1/m)$. But this is not the case. First one does not get a better problem than in the first case, since the $K = \operatorname{diag}(w_b)B$ and σ and τ depend on the condition number of K. If this condition number gets smaller σ and τ can be chosen larger.

Second this would change the optimal value F^* of the problem and therefore x^N and y^N would not be $(1 \pm \varepsilon)$-approximations any more.

During the algorithm the running time of the algorithm is dominated by the two matrix-vector products (vector additions and projections are less expensive). Since every row has only two non-zero entries, it makes sense to use sparse matrices, therefore each matrix-vector product takes $O(m)$ time. Hence the total time per iteration is $O(m)$ which gives total running time of $O(m^2(\varepsilon F^*)^{-1})$ for a $(1+\varepsilon)$-approximation.

Compared with the electrical flow algorithm from previous part the asymptotic runtime is worse. The runtime of the algorithm of Lee is $O(m\sqrt{m}(\varepsilon F^*)^{-1} \log^2(n))$ while the logarithmic terms are neglectable, the difference between m and $\sqrt{m}$ is quite drastic.

But as we will see later, on typical instances in computer vision, the primal-dual algorithm clearly outperforms both electrical flow algorithms.

6. Graph cuts via the ROF model

In the previous chapters the graph cut problem was solved directly by minimizing the objective function.

In this section a more 'indirect' approach is discussed. As we will see later the graph cut problem is closely related to the very famous image denoising model of Rudin, Osher and Fatemi, abbreviated as ROF model or ROF problem.

By solving a such a ROF problem one can get solutions for a whole class of graph cut problems. This approach will be covered in this chapter.

6.1. Solving Graph Cuts via the ROF Model

The Rudin-Osher-Fatemi Total Variation model [53] (ROF model) is a well known continuous image denoising model. Presented in 1992, it was the first model that uses Total Variation minimization in order to denoise an image. The usage of Total Variation allows solutions with sharp discontinuities, furthermore, the calculation of a numerical solution is possible. At this time, other models with similar properties often were very complex and solutions could only be calculated for special cases.

For this thesis we will only consider the discretized versions of the ROF model. For a detailed introduction of the continuous model and its properties see [18, 12].

The ROF model is a special case of a energy minimization model:

$$\min_u \Phi(u) + \frac{1}{\lambda}\Psi(u, f)$$

where f is the noisy input image and $\lambda \in \mathbb{R}$. The function $\Phi(u)$ is called the *regularization term* and measures on how likely it is that u is an actual image. The function $\Psi(u, f)$ is the data term and ensures that the solution does not differ too much from the input image.

One version of the discretized ROF model is:

$$\min_{u \in \mathbb{R}^{M \times N}} \sum_{i,j} \sqrt{(u_{i,j} - u_{i+1,j})^2 + (u_{i,j} - u_{i,j+1})^2} + \frac{1}{2\lambda}\|u - f\|_F^2$$

Where $\|\cdot\|_F$ is the Frobenius norm, i.e. $\|x\|_F := \sqrt{\sum x_{i,j}^2}$. Since, this formulation is not differentiable the s.c. anisotropic Total Variation, given a

$$\min_{u \in \mathbb{R}^{M \times N}} \sum_{i,j} |u_{i,j} - u_{i+1,j}| + |u_{i,j} - u_{i,j+1}| + \frac{1}{2\lambda}\|u - f\|_F^2$$

is used. Or written in matrix-vector form

$$\min_u \|\nabla u\|_1 + \frac{1}{2\lambda}\|u - f\|_F^2 \tag{6.1}$$

where ∇ is the backward-difference operator.

In the following the images will not be seen as elements from the space $\mathbb{R}^{M\times N}$ but as elements from space $\mathbb{R}^{MN}$. In particular this allows to use the euclidean norm instead of the Frobenius norm.

Now, with the same notation and properties for J as in section 4.4 a slightly more abstract version (called the *abstract ROF model*) of this problem is considered:

$$\min_{x\in\mathbb{R}^n} J(y) + \frac{1}{2\lambda}\|G - x\|_2^2 \tag{ROF}$$

In the rest of this section the link between the problems (ROF) and the graph cut problem (P_s), from section 4.4, is established. The original proof is from Chambolle [14]. Recall that ($\bar{P}_s$) denotes the relaxed version of problem (P_s).

Proposition 2 (Chambolle [14] Proposition 3)**.**

(i) Let x be a solution of (ROF) *then $[x > s]$ and $[x \geq s]$ is a solution of* ($\bar{P}_s$)

(ii) Let x be a solution of ($\bar{P}_s$) *and w a solution of* (ROF)*, then*

$$x_i = 1 \Rightarrow w_i \geq s$$
$$x_i = 0 \Rightarrow w_i \leq s$$

In particular this proposition allows to solve the problems ($\bar{P}_s$) for all $s \in \mathbb{R}$ by solving the problem (ROF). On the other hand given the solutions of ($\bar{P}_s$) for all s the solution of (ROF) can be deduced. While solving the problem for all s is not practically, Chambolle showed how to get some approximation if one allows some restrictions of the ROF Model.

For the proof of Proposition 2 another result is needed

Proposition 3 (Chambolle [14], Lemma 1)**.** *Consider two problems of the form* ($\bar{P}_s$)*:*

(i) $x^* = \arg\min_{x\in[0,1]^n} \lambda J(x) + \langle s - G^1, x\rangle$

(ii) $y^* = \arg\min_{x\in[0,1]^n} \lambda J(x) + \langle s - G^2, x\rangle$

with $G_i^1 \geq G_i^2$, $i = 1, \ldots, n$. Then $x_i^* \geq y_i^*$, $i = 1, \ldots, n$.

The proof of this lemma requires further properties of J and is therefore omitted in this thesis, see again [14].

Notation 6. In the next proof problems of the form (P_s) with different values for s will be considered, therefore, for a $t \in \mathbb{R}$ the term (P_t) simply means substituting t for s in Problem (P_s).

Sketch of proof. (of Proposition 2) First a simple fact about the solutions of $(\bar{P}_s)$ for different s. Let $s > s'$ and x^* be a solution of $(\bar{P}_s)$ and y^* a solution of $(\bar{P}_{s'})$ then

$$(P_{s'}) = \min_{x\in[0,1]^n} \lambda J(x) + \langle s - \underbrace{(s - s' + G)}_{:=G'}, x\rangle$$

and it follows

$$s > s' \Longrightarrow G' > G \overset{Prop.3}{\Longrightarrow} y^* \geq x^*$$

Now lets define $w = (w_1, \ldots, w_n)$ as follows

$$x_i := \sup\left\{t \in \mathbb{R} \middle| \exists z \text{ solution of } (P_t) \text{ with } z_i = 1\right\}$$

In order to prove (i) it has to be shown that $[x > s]$ is a solution of (P_s) for all s:

(a) $x_i < s$ then there is no solution y of (P_s) with $y_i = 1$, or in other words for all solutions of (P_s) it follows that $y_i = 0$.

(b) Let $x_i > s$ then one can find a v such that $w_i > v > s$ where (P_v) has a solution y with $y_i = 1$ (since x_i is a supremum v can be arbitrarily close to w_i) since $v > s$ it follows that $x \geq y$ for all y solutions of (P_s) in particular $x_i \geq y_i = 1$.

If $x_i = s$, than also $[x \geq s]$ is a solution.

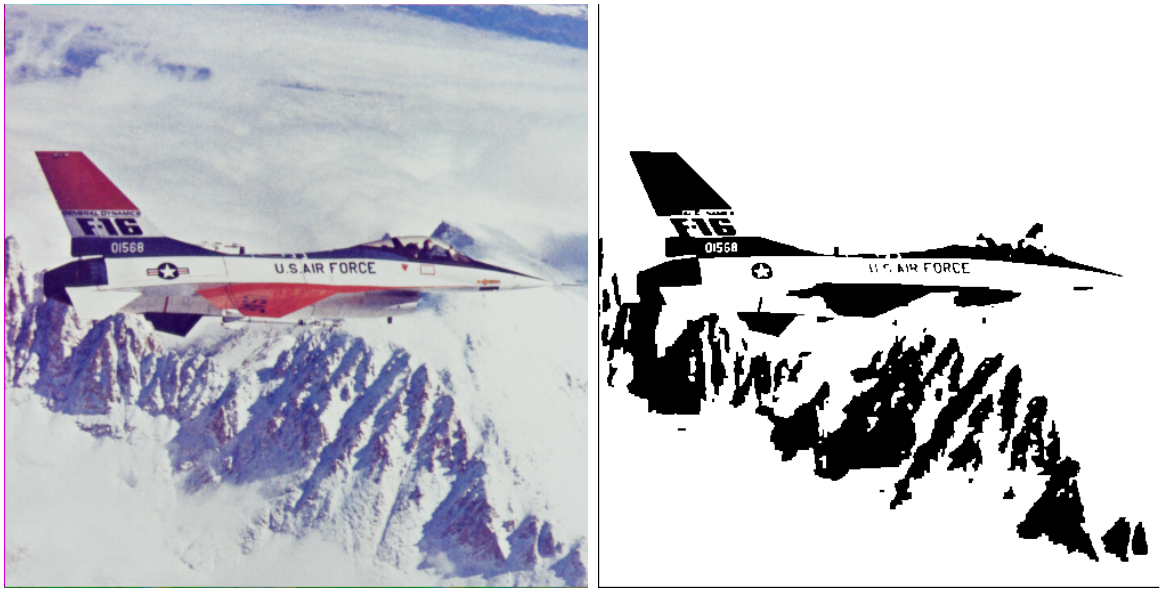
Now for (ii) it has to be shown that w is a solution of (ROF):
For a $x \in \mathbb{R}^n$ let $s' := \min_i x_i$, now the following relation is given:

$$\int_{s'}^{\infty} (s - G_i)[x_i > s]ds = \int_{s'}^{x_i} s - G_i ds = \frac{1}{2}\left((x_i - G_i)^2 - (s' - G_i)^2\right)$$

writting this relation in vector notation and adding $\lambda J(x)$ (recall that $J(x) = 0$ if $x_i = 0$ for all i) yields:

$$\begin{aligned}
\int_{s'}^{\infty} \lambda J([x > s]) + \langle s - G, [x > s]\rangle ds \quad &= \\
\lambda J(x) \quad + \int_{s'}^{\infty} \langle s - G, [x > s]\rangle ds \quad &= \\
\lambda J(x) \quad + \frac{1}{2}\|x - G\|_2 - \frac{1}{2}\|s' - G\|_2 &
\end{aligned}$$

Since we assumed that (i) holds, $[x > s]$ solves (P_s) for all s this means that the left-hand side is minimized, therefore, the right-hand side is minimized, but this is exactly the problem (ROF) (plus some constant term). □



(a) Input image (b) Graph cut solution

(c) ROF solution

Figure 6.1.: Solving a graph cut problem via ROF model

The problem (P_0) is case equal to the other problem formulations given earlier. One advantage of this problem formulation is that the objective function is uniformly convex, this allows the use of the special version of the primal-dual algorithm with faster convergence. Furthermore, another, very fast, method for solving ROF model is given later in this thesis in Section 6.3. Of course on the other hand it is clear that solving the ROF problem is not easier than solving a graph cut.

6.2. Primal-dual for ROF problem

The primal-dual algorithm is well suited for minimizing problems like (ROF) or one of its concrete version, like the problem in (6.1). Note that in the classical ROF problem the underlying graph is 4-connected and all binary weights are 1. Since the previous results hold for the abstract ROF model one can easily use this method for arbitrary weighted graphs by defining $J(x) := \left\|\mathrm{diag}(w^b)Bx\right\|_1$.

Formulating the ROF Problem as a saddlepoint problem by defining:

$$F(Kx) := J(x) \qquad G(x) := \frac{1}{2\lambda}\|x - w^u\|_2$$

The primal-dual algorithm requires the proximal maps of F^* and G. Since F is the same as in the (5.1) the proximal map is again a projection onto $[-w^b, w^b]$

The proximal map of G is given as:

$$\mathrm{prox}_{\tau G}(x) = \frac{\lambda x + \tau w^u}{\lambda + \tau}$$

Hence, the update steps are

$$\begin{cases} y^{k+1} = \mathcal{P}_{[-w^b, w^b]^n}(y^k + \sigma B(2x^k - x^{k-1})) \\ x^{k+1} = {}^{(\lambda x^k - \lambda\tau(B^t y^{k+1} + w^u) + \tau w^u)}/_{(\lambda+\tau)} \end{cases}$$

Furthermore G is uniformly convex with parameter $\frac{1}{\lambda}$ which allows the usage of the faster primal-dual algorithm, given in Algorithm 11.

Definition 40. (uniformly convex)
A convex function f is called uniformly convex with parameter μ if $\forall x, y$ of the domain of f and $\forall \lambda \in [0, 1]$

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y) - \lambda(1-\lambda)\mu(\|x - y\|)$$

holds.
This is a generalization of the concept of strong convexity.

This algorithm has a convergence rate of $O({}^1/_{k^2})$ which is a large improvement over the convergence rate of $O({}^1/_k)$ of the standard algorithm.

6.3. Block coordinate descent and 1D TV minimization

In the following section another approach for solving the ROF problem will be discussed. This approach uses block coordinate descent, which is a well known optimization technique, see for instance [2]. The main idea of block coordinate descent is to split up the variables into blocks and optimize each block individually.

Algorithm 11 Primal-dual algorithm for uniformly convex F or G

choose $\sigma_0, \tau_0 > 0$, $\sigma_0 \tau_u L^2 \leq 1$
choose $(x^0, y^0) \in X \times Y$
$\bar{x}^0 = x^0$, $k = 0$
while not converged **do**
    $y^{k+1} = (I + \sigma_n \partial F^*)^{-1}(y^k + \sigma_k K \bar{x}^k)$
    $x^{k+1} = (I + \tau_k \partial G)^{-1}(x^k - \tau_k K^t y^k)$
    $\theta_k = 1/\sqrt{1+\gamma\tau_k}$
    $\tau_{k+1} = \theta_k \tau_k$
    $\sigma_{k+1} = \sigma_k/\theta_k$
    $\bar{x}^{k+1} = x^{k+1} + \theta_k(x^{k+1} - x^k)$
    $k \leftarrow k + 1$

The following method for solving the ROF Problem is based on the work of Chambolle and Pock [16].

If the underling graph is a 4-connected image, the ROF model, in terms of total variation can be written as

$$\min_u \lambda J(u) + \frac{1}{2}\|G - u\|_2^2 = \min_u \lambda TV(u) + \frac{1}{2}\|G - u\|_2^2 \tag{6.2}$$

Furthermore, only the case $\lambda = 1$ will be considered since for graph cuts only this case is of interest.

Recall that the anisotropic version of the total variation is given as

$$TV(u) := \sum_{i,j} |u_{i+1,j} - u_{i,j}| + |u_{i,j+1} - u_{i,j}|$$

This allows to split up into the horizontal and the vertical total variation:

$$\mathrm{TV_h}(u) := \sum_{i,j} |u_{i,j+1} - u_{i,j}| = \sum_i \sum_j |u_{i,j+1} - u_{i,j}|$$
$$\mathrm{TV_v}(u) := \sum_{i,j} |u_{i+1,j} - u_{i,j}| = \sum_j \sum_i |u_{i+1,j} - u_{i,j}|$$

which allows us to rewrite problem in (6.2) as:

$$\min_u \mathrm{TV_h}(u) + \mathrm{TV_v}(u) + \frac{1}{2}\|G - u\|_2^2$$

and introducing two additional variables for u, the following constrained problem is obtained:

$$\min_{u,u_1,u_2} \mathrm{TV_h}(u_1) + \mathrm{TV_v}(u_2) + \frac{1}{2}\|G - u\|_2^2$$
$$s.t. \quad u_1 = u, u_2 = u$$

The standard technique for solving such a constrained optimization problem is to add Lagrangian multipliers (p_1 and p_2), which will transform the problem into:

$$\max_{p_1,p_2} \min_{u,u_1,u_2} \mathrm{TV_h}(u_1) + \mathrm{TV_v}(u_2) + \frac{1}{2}\|G-u\|_2^2 + \langle u-u_1, p_1\rangle + \langle u-u_2, p_2\rangle \tag{6.3}$$

Now it can be observed that the terms containing u_1 do not depend on terms containing u_2 which allows to group them and minimize separately.

Furthermore, the minimization problem for u_1 is the convex conjugate of the total variation, i.e.

$$\min_{u_1} \mathrm{TV_h}(u_1) - \langle u_1, p_1\rangle = -\max_{u_1} \langle u_1, p_1\rangle - \mathrm{TV_h}(u_1) = -\mathrm{TV_h^*}(p_1)$$

and analog for u_2, p_2. This allows to write (6.3) as:

$$\max_{p_1,p_2} \min_{u} -\mathrm{TV_h^*}(p_1) - \mathrm{TV_v^*}(p_2) + \langle u, p_1+p_2\rangle + \frac{1}{2}\|u-G\|_2^2$$

Minimizing for u is simple, since the optimality condition is given by

$$(p_1+p_2) + u - G = 0$$

and expressing u in terms of p_1, p_2 and G the problem becomes

$$\max_{p_1,p_2} -\mathrm{TV_h^*}(p_1) - \mathrm{TV_v^*}(p_2) + \langle G, p_1+p_2\rangle - \frac{1}{2}\|p_1+p_2\|_2^2$$

By changing from the maximization problem to the minimization problem (by multiplying with -1) and adding a constant factor of $\frac{1}{2}\|G\|_2^2$ on gets:

$$\min_{p_1,p_2} \mathrm{TV_h^*}(p_1) + \mathrm{TV_v^*}(p_2) + \frac{1}{2}\|(p_1+p_2)-G\|_2^2$$

Compared to the original problem this one does not has any constraints.

Using the idea of coordinate descent the idea is now to alternately minimize for p_1 and p_2, i.e. for p_1 (and analog for p_2)

$$p_1^{new} = \arg\min_{p_1} \underbrace{\mathrm{TV_h^*}(p_1)}_{:=f(p_1)} + \underbrace{\frac{1}{2}\|p_1 + (p_2-G)\|_2^2}_{:=g(p_1)} + \underbrace{\mathrm{TV_v^*}(p_2)}_{:=c} \tag{6.4}$$

since the last term is constant it can be dropped.

The general method of splitting the variables into blocks and optimize each of these blocks alternatively converges with sublinear rate of $O(1/n)$[2].

But first one wants to get rid of the dual of the TV, or at least do not explicitly calculate it. Problems of the form $\min_x f(x) + g(x)$ (where f convex, g smooth, convex and with Lipschitz continuous gradient L) can be solved by the s.c. proximal gradient algorithm [63]. The iterates of this algorithm are very simple and given by:

$$x^{k+1} = \mathrm{prox}_{tf}(x^k - t\nabla g(x^k))$$

this algorithm converges for $t \in (0, 2/L)$. The advantage of an iteration step of this form is that, with the help of the Moreau identity, one can circumvent the explicit calculation of the convex conjugate of the total variation:

$$\operatorname{prox}_{\mathrm{TV}_\mathrm{h}^*}(x) = x - \operatorname{prox}_{\mathrm{TV}_\mathrm{h}}(x) = x - \left(\arg\min_y \frac{1}{2}\|x-y\|_2^2 + \mathrm{TV}_\mathrm{h}(y) \right)$$

The gradient of problem (6.4) is $\nabla g(p_1) = p_1 + p_2 - G$ and its Lipschitz constant is 1, therefore, an iteration step of the proximal point algorithm is given by:

$$p_1^{k+1} = (G - p_2^k) - \arg\min_y \left[\frac{1}{2} \left\| y + p_2^k - G \right\|_2^2 + \mathrm{TV}_\mathrm{h}(y) \right]$$

As one easily can observe this update step now is again a ROF problem.

The whole procedure for alternatively optimize for p_1 and p_2 is shown in Algorithm 12.

Algorithm 12 Block coordinate for the ROF problem

1: $k = 0$, choose p_1^0, p_2^0
2: **while** not converged **do**
3: $\quad p_1^{k+1} = (G - p_2^k) - \arg\min_y \left[\frac{1}{2} \left\| y + p_2^k - G \right\|_2^2 + \mathrm{TV}_\mathrm{h}(y) \right]$
4: $\quad p_2^{k+2} = (G - p_1^k) - \arg\min_y \left[\frac{1}{2} \left\| y + p_1^k - G \right\|_2^2 + \mathrm{TV}_\mathrm{v}(y) \right]$
5: $\quad k \leftarrow k + 1$
return $u \leftarrow G - (p_1^k + p_2^k)$

Moreover these update steps in line 3 and 4 can be simplified further since:

$$\begin{aligned} &\arg\min_y \frac{1}{2}\|y + p_2 - G\|_2^2 + \mathrm{TV}_\mathrm{h}(y) = \\ &\arg\min_y \sum_{i,j} \frac{1}{2}(y_{i,j} + (p_2)_{i,j} - G_{i,j})^2 + \sum_{i,j} |y_{i,j+1} - y_{i,j}| = \\ &\sum_i \arg\min_y \sum_j \frac{1}{2}(y_{i,j} + (p_2)_{i,j} - G_{i,j})^2 + |y_{i,j+1} - y_{i,j}| \end{aligned} \tag{6.5}$$

Equation (6.5) indicates that the update steps are now reduced to multiple independent 1-dimensional ROF problems. These independent problems can easily be solved in parallel.

The 1-dimensional ROF problem is a special case of of the s.c. Fused Lasso Signal Approximator (FLSA) [26] and there exists various methods for this problem, see for instance [31, 21].

The proximal gradient algorithm has a convergence rate of $O(\frac{1}{k})$ but there exists a accelerated version called FISTA [3] which has a convergence rate of $O(\frac{1}{k^2})$ and as shown

Algorithm 13 Accelerated block coordinate for the ROF problem

$k = 0$, choose: p_1^0, $p_2^0 = p_2^{-1}$, $t_1 = 1$
while not converged **do**
  $t_{k+1} = {}^{(1+\sqrt{1+4(t_k)^2})}/_2$
  $\bar{p}_2^k = p_2^k + {}^{t_k-1}/_{t_{k+1}}(p_2^k - p_2^{k-1})$
  $p_1^{k+1} = (G - \bar{p}_2^k) - \arg\min_y \left[\frac{1}{2}\left\|y + \bar{p}_2^k - G\right\|_2^2 + \mathrm{TV_h}(y)\right]$
  $p_2^{k+1} = (G - p_1^k) - \arg\min_y \left[\frac{1}{2}\left\|y + p_1^k - G\right\|_2^2 + \mathrm{TV_v}(y)\right]$
  $k \leftarrow k + 1$
return $u \leftarrow G - (p_1^k + p_2^k)$

by Chambolle and Pock block coordinate descent methods can also be accelerated in a FISTA like manner, if the problem is splitt into exactly two blocks [16]. This allows to accelerate Algorithm 12 which results in Algorithm 13.

The decomposition into the blocks is arbitrary and not restricted to blocks containing the vertical and horizontal chains. In their paper Chambolle and Pock also presented a version for the ROF model with squares instead of chains.

While convergence of the accelerated algorithm is only guaranteed for two blocks we will later see that decomposing 8-connected graphs into four blocks also yields promising results and all the examples converge.

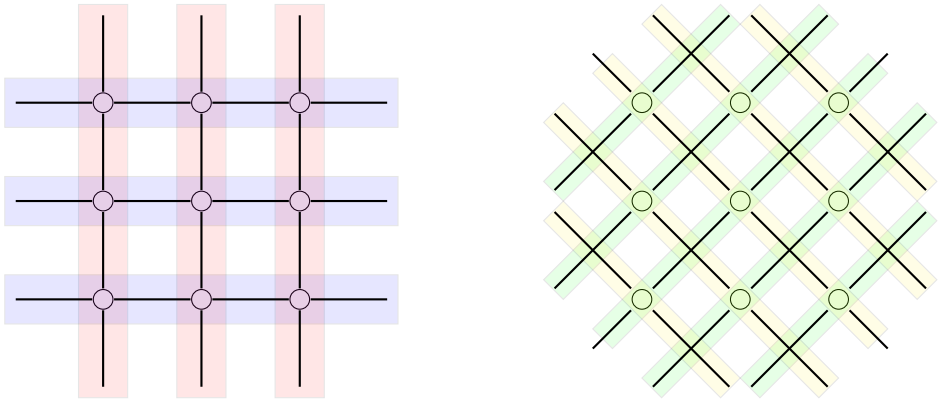
These 8-connected graphs will be decomposed into four blocks. Same as for 4-connected graphs one block will contain all vertical, one block all horizontal weights. Furthermore, two additional blocks one containing the 'northeast' weights and the other containing the 'northwest' weights are used. See figure 6.2 for an illustrations of this decomposition.

Furthermore, experiments for 6-connected graphs, representing three dimensional rectangular boxes, will be performed. These graphs are decomposed into three blocks, each block grouping all weights in a specific direction, i.e. one block for all edges in x-direction, etc.

The Algorithms 12 and 13 only gave the steps for two blocks, now if l blocks are use one get variables $p_1, \ldots, p_l$ and the update steps for this variables, in these Algorithms will become:

$$p_i^{k+1} = \left(G - \sum_{j=1, j\neq i}^{l} p_j^k\right) - \arg\min_y \left[\frac{1}{2}\left\|y + \sum_{j=1, j\neq i}^{l} p_j^k) - G\right\|_2^2 + \mathrm{TV_v}(y)\right]$$

In the accelerated verion the extrapolation step has to be performed for all variables beside p_1.



(a) Vertical and horizontal chains (b) Diagonal chains

Figure 6.2.: The different blocks for the block coordinate descent algorithm

7. Experiments and Discussion

In this last chapter of the thesis the previously described algorithms are evaluated and the results are presented. Afterwards this results are discussed and interpreted.

7.1. Experiments

In this sections the results of the primal-dual algorithm, the global relabeling method and the block coordinate descent algorithm are presented and compared to the algorithm of Boykov and Kolmogorov[8], which is considered one of the fastest algorithm for computer vision problems.

7.1.1. Details

Implementation details

For the experiments Walter Bell's implementation of Boykov's and Kolmogorov's algorithm was used. The code is available at his homepage [4]. This algorithm requires integer weights, therefore, the weights were scaled such they to lie in the interval $[1, 256]$ and rounded.

For the sake of readability the algorithm of Boykov will be denoted by *boy*, the standard primal-dual algorithm is denoted as *pd* and the primal-dual algorithm with global relabeling as *pd-gr*. The block coordinate descent algorithm is denoted by *bc* and its accelerated version as *bc-ac*.

The algorithm are implemented in MATLAB(R) 2013a and the (time) critical parts were implemented in C/C++. The operating system was Debian 7 (wheezy) and as compiler GNU g++ 4.7.2 was used.

The experiments are performed on a computer with two Intel® Xeon® CPU X5680 with 3.33GHZ, therefore in total 12 physical cores could be used.

While the CPU's offer 24 logical cores (and the operating system reports 24 cores) using more than 12 cores does not provide any speedups and actually resulted in an increased runtime.

For parallelization the Open MP library was used.

For the 1-dimensional TV optimization the algorithm of Johnson [31], which has linear runtime, was used.

Input data

For the 4-connected and the 8-connected problem instances binary weights (w^b) and the unary weights (w^s, w^t) were calculated from an input image $f \in [0,1]^{MN}$ with the following functions:

$$w_i^t := a f_i^2$$
$$w_i^s := a(f_i - 1)^2$$
$$w_i^b := \exp(-b\,|(Bf)_i|)$$

where $a = 1$ and $b = 20$. As input some old standard classic test images were used [20]. Since this collection do not contain high resolution images some additional images from [50] were used.

Some tests with 6-connected graphs were performed as well. As input the 'Bunny' dataset used by Lempitsky et al. [43] and the 'Liver' datasets used by various papers of Boykov et al. [9] were used. Both datasets can be found at on the homepage of Computer Vision Research Group at the University of Western Ontario [47].

7.1.2. 4 and 8-connected graphs

In Table 7.1 the number of iterations of the block chain algorithm and the primal-dual algorithms were compared. Table 7.2 compares the running times of the block coordinate descent algorithms to the algorithm *boy* (note that this algorithm is single threaded).

		bc-ac	*bc*	*pd-gr*
birds	192x128	1	1	53
	384x256	1	1	102
	768x512	4	7	102
	1536x1024	4	7	202
pedestrian	192x128	46	169	446
	384x256	28	101	345
	768x512	55	137	421
	1536x1024	-	-	2302
birds	192x128	2	2	60
	384x256	2	2	56
	768x512	3	3	102
	1536x1024	7	13	202
pedestrian	192x128	10	23	202
	384x256	45	311	502
	768x512	65	387	902
	1536x1024	-	4782	1692

Table 7.1.: Comparisons of the number of iterations, the first half of the table are 4-connected graphs and the second half 8-connected, the algorithm was stopped after 5000 iterations

In order to decrease the runtime, the solution of the 2-dimensional ROF problem was only calculated after every 100 iterations. Therefore, the stopping criterion of these algorithms were only evaluated every 100th iteration.

	8 connected					4 connected				
#cores	*boy*	*bc-ac*	*bc*	*pd-gr*	*pd*	*boy*	*bc-ac*	*bc*	*pd-gr*	*pd*
1	0.55	7.80	14.50	56.04	463.36	0.25	3.17	5.87	13.41	76.18
3		3.49	6.58	42.89	362.95		1.14	2.17	13.09	75.90
6		1.99	3.53	41.79	426.47		0.85	1.54	22.74	134.80
12		1.78	2.50	57.51	378.56		0.53	1.01	15.50	84.25

Table 7.2.: Number of cores vs runtime in seconds, the input image had a resolution of 512x512

In Table 7.3 the performance of the global relabeling method for different relabeling intervals is shown.

		I			
		100	max(I,J)/2	max(I,J)	primal-dual
birds	192x128	53	53	53	53
	384x256	102	194	307	307
	768x512	102	265	265	265
	1536x1024	202	770	929	929
	3072x2048	202	1082	1082	1082
pedestrian	192x128	446	482	415	4328
	384x256	345	386	770	1519
	768x512	421	770	1538	2801
	1536x1024	2302	1700	3188	14296
	3072x2048	4602	3074	6146	15810
birds	192x128	60	60	60	60
	384x256	56	56	56	56
	768x512	102	342	342	342
	1536x1024	202	609	609	609
	3072x2048	225	1538	1616	1616
pedestrian	192x128	202	194	194	217
	384x256	502	578	770	4109
	768x512	902	1154	1538	5755
	1536x1024	1692	2306	3074	4139
	3072x2048	1202	4610	6146	12900

Table 7.3.: Table showing the number of iterations until the global relabeling algorithm converges. I is the number of primal-dual steps until the next relabeling step is performed. The rightmost column gives the number of iterates the standard primal-dual algorithm without relabeling needs. The upper half of the table are 4-connected problems and the lower half 8-connected problems.

7.1.3. 6-connected graphs

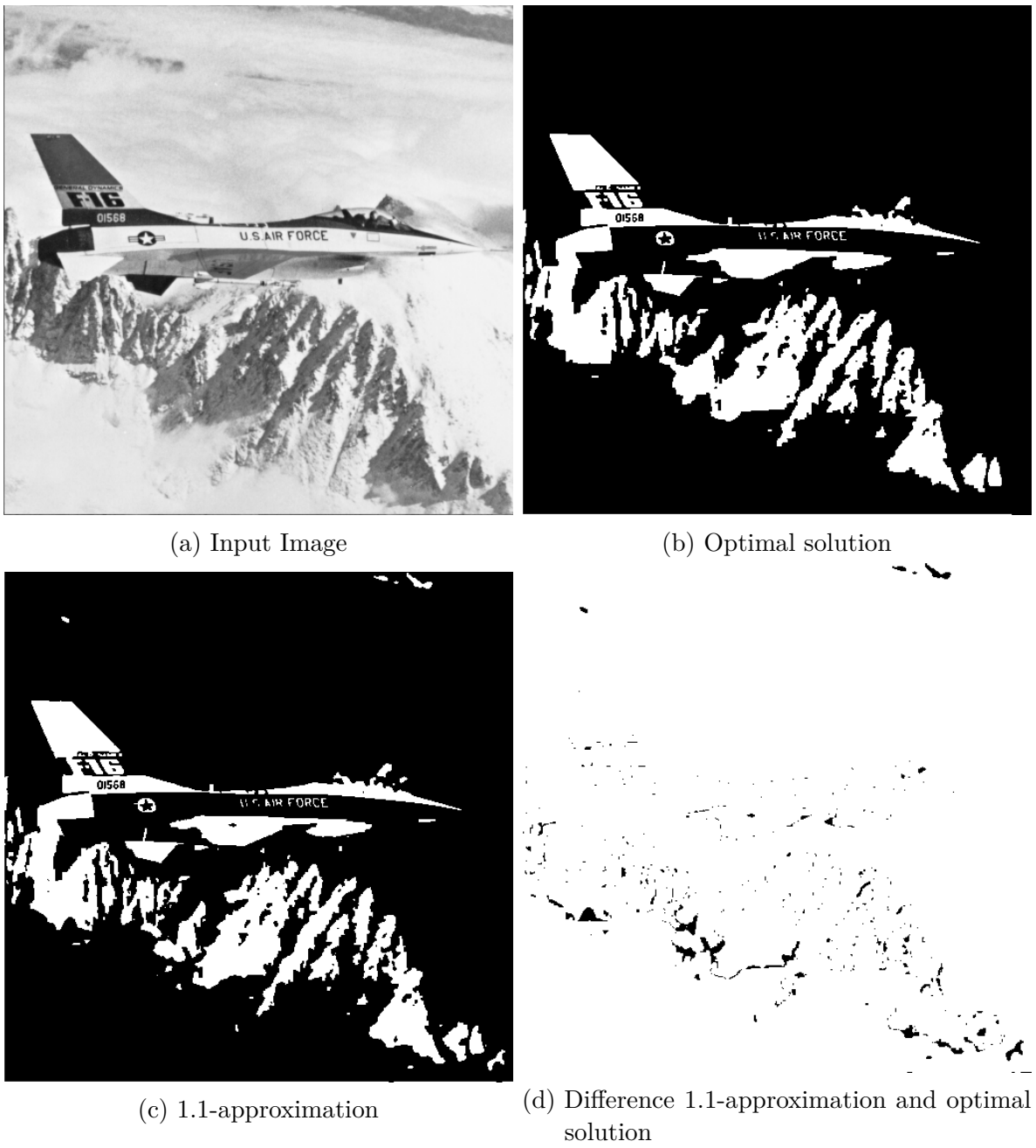
For the small version of the 'bunny' dataset the accelerated block coordinate descent algorithms performed over 26800 iterations until the optimal cut was found. At this time the primal-dual gap of the ROF problem was less then 10^{-15}. But the non accelerated version found the optimal value after 1000 iterations. Nevertheless, the 1000 iterations took around 45 seconds while algorithm *boy* needed a little bit more than one second. For the larger instances both versions of the block coordinate descent algorithm did not find the optimal solution within 40000 iterations.

7.1.4. $(1+\varepsilon)$-Approximations

In Section 5.4 a bound of the number of iterations the primal-dual algorithms needs for a $(1+\varepsilon)$ approximation. As seen in table 7.4 this bound is very crude for computer vision instances. Nevertheless, table 7.4 shows that the primal-dual algorithm only needs very few iterations in order to calculate a $(1+\varepsilon)$-approximate solution for reasonable ε values.

		ε			
		0.1	0.01	0.001	N
airplane	16x16	6	37	55	$29.9176 \cdot \varepsilon^{-1}$
	32x32	6	51	101	$51.2702 \cdot \varepsilon^{-1}$
	64x64	8	36	81	$53.0279 \cdot \varepsilon^{-1}$
	128x128	9	39	69	$53.6555 \cdot \varepsilon^{-1}$
	256x256	9	40	98	$53.5296 \cdot \varepsilon^{-1}$
	512x512	10	46	151	$53.4626 \cdot \varepsilon^{-1}$

Table 7.4.: Number of iterations for different ε. The value in the last column is the theoretical bound from Section 5.4 (for the different sizes the cut values are different, which explains why the theoretical bounds are nearly the same)



(a) Input Image

(b) Optimal solution

(c) 1.1-approximation

(d) Difference 1.1-approximation and optimal solution

Figure 7.1.: $(1+\varepsilon)$-Approximation Example, for $\varepsilon = 0.1$

7.2. Discussion

In this section the experimental results from the previous section are discussed. First some notes are given about the primal-dual algorithm and global relabeling. Afterwards the approach of solving graph cuts via the ROF model are discussed and in particular some numerical problems with block coordinate decent approach are addressed.

7.2.1. Primal-Dual algorithm and Global Relabeling

As one can clearly see from table 7.3 global relabeling provides a huge improvement over the standard primal-dual algorithm regarding iterations and of course also the runtimes are much better as table 7.2 indicates.

During the experiments a relabeling step was performed every 100 iterations, since this offered better convergence speed, see table 7.3. If relabeling steps are performed after $\max(M, N)$ iterations, for some resolutions, the algorithm already fully converged before the first relabeling step takes place.

In general the global relabeling algorithm can not compete with the highly optimized algorithm *boy*. The reason why the primal-dual algorithm is extensively used in computer vision is due to the fact that it can easily be parallelized on GPUs which offers a huge speed-up compared to a standard CPU implementation. For a discussion see for instance [62].

7.2.2. The ROF model and block coordinate descent

One problem of this approach is that one has no control over the cut energy for a specific value of s. In other words while the primal-dual gap of the ROF problem is decreasing no information about the energy of the cut value (beside thresholding and calculating manually) is known.

Furthermore, the rate of convergence of the cut energy is unknown, only if the ROF problem is solved exactly the cut is solved exactly. While the current ROF solution can be close to the optimal solution, the cut obtained by thresholding can be arbitrarily large. Figure 7.2 illustrates this property.

Especially entries of the ROF solution close to 0 are critical since a small change in the ROF solution could severely change the cut solution. Since, for practical reasons, all calculation are performed with finite precision floating point numbers, this problem is hard to come by.

The plots in Figure 7.3 illustrate this fact by showing how the energy of the cut can advance while solving the ROF model on the left side. On the right side the number of labels changing from one partition to the other is shown.

The first example is a 'good' one, the energy of the cut is slowly decreasing until the optimum is reached. The energy in the second example again decreasing fast but then for over a quarter of the total iterations, nothing happens. Between iteration 1250 and iteration 2000 the solution does not change but the optimal value is not reached ether. Contrary in the third example the optimum is found quite fast and shortly before reaching the optimal solution a lot of change happens. The last example is a combination between the second and third example. At the beginning fast convergence can be observed, afterwards there is no change and at the end a lot of labels are changing.

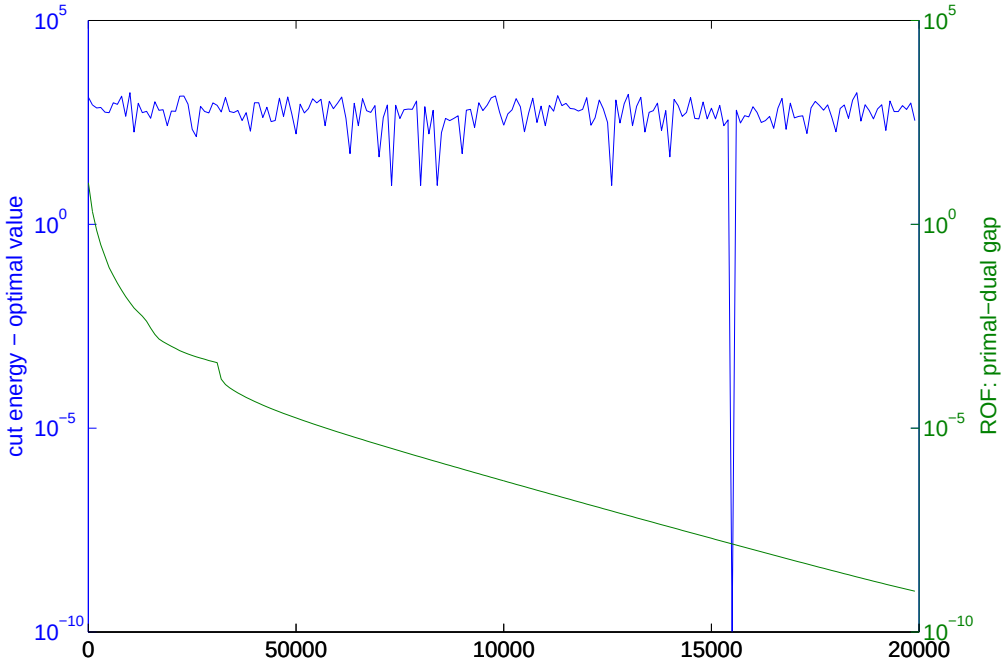


Figure 7.2.: Difference between cut energy and the optimal value and the primal-dual gap of the ROF problem, note that the optimal solution would be found at iteration 15600 but is lost afterwards, nevertheless the gap of the ROF problem gets decreased

Block Coordinate Descent

The ROF problem is always solved much faster by the accelerated version of the Block Coordinate Descent algorithm. In a few rare cases it happens that the non-accelerated algorithm finds the optimal cut faster than the accelerated algorithm.

In general, at the beginning of the algorithm, the cut value is decreasing faster for the accelerated version, but the optimal value is not exactly reached. This is probably due to the fact that the ROF gap is decreased very fast but afterwards the changes of the ROF solution are getting very small, therefore it is unlikely for some variable to change from being larger to being smaller than the threshold value.

For 4 and 8-connected graphs the Kahan summation algorithm[32] had to be used for the calculation of the primal and dual energy. Without this summation algorithm the primal-dual gap was severely wrong and even got negative. As seen in table 7.1 there are again instances which are solved immediately and instances which are not solved after 5000 iterations.

Since the 6-connected problem instances are much larger (5 million vertices) these instances are more sensible to numerical errors. This is further illustrated by the fact, that the difference between the primal energy, calculated with the Kahan summation

algorithm and without can be as large as 1.

Nevertheless, as seen in table 7.2, for smaller instances the block coordinate descent methods are very fast. For the 4-connected problem the runtime of *boy* is twice as fast and there clearly is hope that with around 24 physical cores the runtimes are nearly the same.

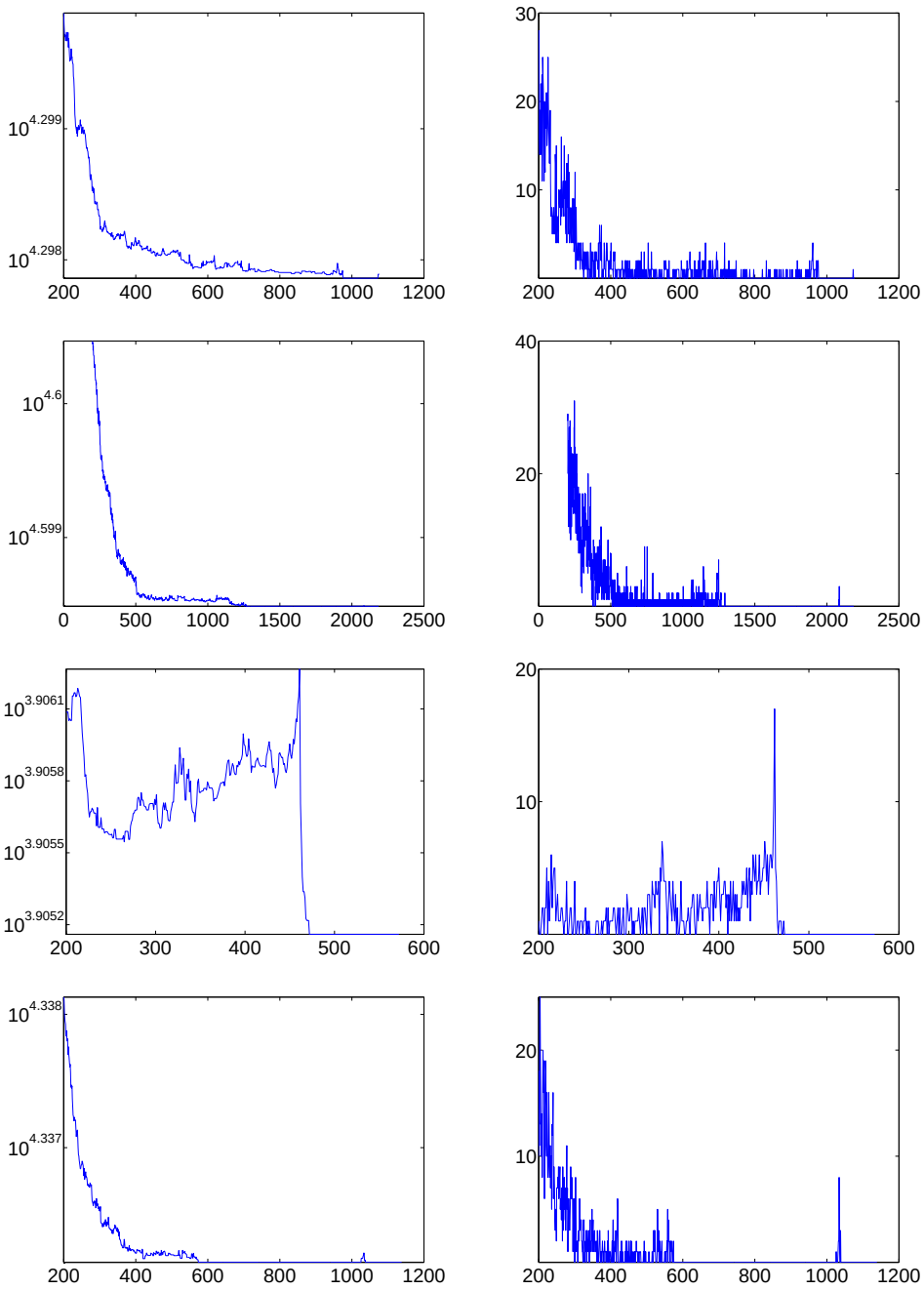


Figure 7.3.: On the left the cut values during the algorithm and on right side the number of labels changing from one partition into the other. Note that the values for first two hundred iterations are not ploted. For all four examples the input was a 4-connected 512x512 grid graph

Conclusion

In the first part two algorithms for approximating graph cuts were presented. While these algorithms have a superior asymptotic running time, the experiments showed that even for very small computer vision instances the computations are very time-consuming and can not compete with other algorithms. This indicates that the runtime constants are very high and the used problem sizes were to small. For increased problem sizes these algorithms may become more competitive. Another drawback of the algorithm of Lee et al. is that the algorithm is only suitable for problems where all weights are set to the same value, which is a very strong restriction.

In the second part the primal-dual algorithm of Chambolle and Pock was presented and it was shown that global relabeling clearly improves the performance of this algorithm. Furthermore, it was shown how one can obtain a graph cut solution by solving the Rudin-Osher-Fatemi functional. Solving this functional with accelerated coordinate descent method, which can be parallelized easily, is a competitive approach compared to state of the art graph cut algorithms for computer vision problems. Unfortunately, for very large problem instances numerical instabilities could be observed.

Bibliography

[1] S. Arora, E. Hazan, and S. Kale. "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications." In: *Theory of Computing* 8.1 (2012), pp. 121–164.

[2] A. Beck and L. Tetruashvili. "On the Convergence of Block Coordinate Descent Type Methods". In: *SIAM Journal on Optimization* 23.4 (2013), pp. 2037–2060.

[3] A. Beck and M. Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM Journal on Imaging Sciences* 2.1 (2009), pp. 183–202.

[4] W. Bell. *A C++ implementation of a Max Flow-Graph Cut algorithm*. URL: http://www.cs.cornell.edu/vision/wbell (visited on 01/20/2015).

[5] A. A. Benczúr and D. R. Karger. "Approximating st minimum cuts in $\tilde{O}$ (n 2) time". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM. 1996, pp. 47–55.

[6] A. A. Benczúr and D. R. Karger. "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs". In: *SIAM Journal on Computing* 44.2 (2015), pp. 290–319.

[7] B. Bollobás. *Modern graph theory*. Vol. 184. Springer, 1998.

[8] Y. Boykov and V. Kolmogorov. "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.9 (2004), pp. 1124–1137.

[9] Y. Boykov and V. Kolmogorov. "Computing Geodesics and Minimal Surfaces via Graph Cuts." In: *ICCV*. IEEE Computer Society, 2003, pp. 26–33.

[10] Y. Boykov and V. S. Lempitsky. "From Photohulls to Photoflux Optimization." In: *BMVC*. Ed. by M. J. Chantler, R. B. Fisher, and E. Trucco. British Machine Vision Association, 2006, pp. 1149–1158.

[11] Y. Boykov, O. Veksler, and R. Zabih. "Markov Random Fields with Efficient Approximations." In: *CVPR*. IEEE Computer Society, 1998, pp. 648–655.

[12] K. Bredies and D. A. Lorenz. *Mathematische Bildverarbeitung - Einführung in Grundlagen und moderne Theorie*. Vieweg+Teubner, 2011, pp. I–X, 1–445.

[13] R. E. Burkard and U. Zimmermann. *Einführung in die Mathematische Optimierung*. Springer, 2012.

[14] A. Chambolle. "Total variation minimization and a class of binary MRF models". In: *Energy minimization methods in computer vision and pattern recognition*. Springer. 2005, pp. 136–152.

[15] A. Chambolle and T. Pock. "A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging". In: *Journal of Mathematical Imaging and Vision* 40.1 (2011), p. 120.

[16] A. Chambolle and T. Pock. "A remark on accelerated block coordinate descent for computing the proximity operators of a sum of convex functions." to be published. 2015.

[17] A. Chambolle and T. Pock. "On the ergodic convergence rates of a first-order primal-dual algorithm". to be published. 2014.

[18] A. Chambolle et al. "An introduction to Total Variation for Image Analysis". In: *Theoretical Foundations and Numerical Methods for Sparse Recovery, De Gruyter, Radon Series Comp. Appl. Math.* 9 (2010), pp. 263–340.

[19] P. Christiano et al. "Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs". In: *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*. STOC '11. San Jose, California, USA: ACM, 2011, pp. 273–282.

[20] *Classic Test Images*. URL: http://www.hlevkin.com/TestImages/ (visited on 01/20/2015).

[21] L. Condat. "A Direct Algorithm for 1-D Total Variation Denoising". In: *Signal Processing Letters, IEEE* 20.11 (2013), pp. 1054–1057.

[22] J. Edmonds and R. M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." In: *J. ACM* 19.2 (1972), pp. 248–264.

[23] P. Elias, A. Feinstein, and C. E. Shannon. "A note on the maximum flow through a network." In: *IRE Transactions on Information Theory* 2.4 (1956), pp. 117–119.

[24] L. R. Ford and D. R. Fulkerson. "Maximal Flow through a Network." In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404.

[25] M. L. Fredman and R. E. Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms." In: *J. ACM* 34.3 (1987), pp. 596–615.

[26] J. Friedman et al. "Pathwise coordinate optimization". In: *Annals of Applied Statistics 2007, Vol. 1, No. 2, 302-332* (2007).

[27] A. V. Goldberg and S. Rao. "Beyond the flow decomposition barrier". In: *Journal of the ACM (JACM)* 45.5 (1998), pp. 783–797.

[28] A. V. Goldberg and R. E. Tarjan. "A new approach to the maximum-flow problem". In: *J. ACM* 35.4 (1988), pp. 921–940.

[29] A. V. Goldberg et al. "Maximum Flows by Incremental Breadth-First Search". In: *Algorithms – ESA 2011*. Springer, 2011.

[30] D. M. Greig, B. T. Porteous, and A. H. Seheult. "Exact Maximum A Posteriori Estimation for Binary Images". English. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 51.2 (1989), pp. 271–279.

[31] N. A. Johnson. "A Dynamic Programming Algorithm for the Fused Lasso and L 0-Segmentation". In: *Journal of Computational and Graphical Statistics* 22.2 (2013), pp. 246–260.

[32] W. Kahan. "Pracniques: further remarks on reducing truncation errors." In: *Commun. ACM* 8.1 (1965), p. 40.

[33] D. R. Karger. "Random sampling in cut, flow, and network design problems." In: *STOC*. Ed. by F. T. Leighton and M. T. Goodrich. ACM, 1994, pp. 648–657.

[34] D. R. Karger. "Using Randomized Sparsification to Approximate Minimum Cuts." In: *SODA*. Ed. by D. D. Sleator. ACM/SIAM, 1994, pp. 424–432.

[35] J. A. Kelner et al. "An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations". In: *arXiv preprint arXiv:1304.2338* (2013).

[36] V. Kolmogorov and R. Zabih. "Computing visual correspondence with occlusions using graph cuts". In: *IEEE ICCV*. 2001, pp. 508–515.

[37] V. Kolmogorov and R. Zabih. "What energy functions can be minimized via graph cuts?" In: *IEEE TPAMI* 26.2 (2004), pp. 147–59.

[38] I. Koutis, G. Miller, and R. Peng. "A Nearly-m log n Time Solver for SDD Linear Systems". In: *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE, 2011, pp. 590–598.

[39] I. Koutis, G. Miller, and R. Peng. "Approaching Optimality for Solving SDD Linear Systems". In: *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*. IEEE, 2010, pp. 235–244.

[40] I. Koutis, G. L. Miller, and R. Peng. "Approaching Optimality for Solving SDD Linear Systems". In: *SIAM Journal on Computing* 43.1 (2014), pp. 337–354.

[41] *Lean Algebraic Multigrid (LAMG) MATLAB Software*.

[42] Y. T. Lee, S. Rao, and N. Srivastava. "A new approach to computing maximum flows using electrical flows". In: *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*. ACM. 2013, pp. 755–764.

[43] V. S. Lempitsky and Y. Boykov. "Global Optimization for Shape Fitting." In: *CVPR*. IEEE Computer Society, 2007.

[44] V. S. Lempitsky, Y. Boykov, and D. V. Ivanov. "Oriented Visibility for Multiview Reconstruction." In: *ECCV (3)*. Ed. by A. Leonardis, H. Bischof, and A. Pinz. Vol. 3953. Lecture Notes in Computer Science. Springer, 2006, pp. 226–238.

[45] A. Madry. "Navigating Central Path with Electrical Flows: from Flows to Matchings, and Back." In: *CoRR* abs/1307.2205 (2013).

[46] D. Martin et al. "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics". In: *Proc. 8th Int'l Conf. Computer Vision*. Vol. 2. 2001, pp. 416–423.

[47] *Max-flow problem instances in vision*. URL: http://vision.csd.uwo.ca/data/maxflow/ (visited on 01/20/2015).

[48] J. B. Orlin. "Max flows in O (nm) time, or better". In: *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*. ACM. 2013, pp. 765–774.

[49] S. A. Plotkin, D. B. Shmoys, and É. Tardos. "Fast Approximation Algorithms for Fractional Packing and Covering Problems". In: *FOCS*. IEEE Computer Society, 1991, pp. 495–504.

[50] *Public doman archive*. http://publicdomainarchive.com/wp-content/uploads/2014/02/public-domain-images-free-high-quality-resolution-2014-02-25-0008.jpg and http://publicdomainarchive.com/wp-content/uploads/2014/02/public-domain-images-free-high-resolution-quality-photos-unsplash-0218.jpg. URL: http://publicdomainarchive.com/ (visited on 01/20/2015).

[51] R. T. Rockafellar. *Convex analysis*. Princeton Mathematical Series. Princeton, N. J.: Princeton University Press, 1970.

[52] C. Rother, V. Kolmogorov, and A. Blake. ""GrabCut": interactive foreground extraction using iterated graph cuts." In: *ACM Trans. Graph.* 23.3 (2004), pp. 309–314.

[53] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms". In: *Physica D: Nonlinear Phenomena* 60.1-4 (1992), pp. 259–268.

[54] W. Rudin. *Real and Complex Analysis*. McGraw-Hill Science/Engineering/Math, 1986.

[55] D. Scharstein and R. Szeliski. "High-accuracy stereo depth maps using structured light". In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2003, pp. I–195.

[56] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

[57] S. Seitz et al. "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2006, pp. 519–528.

[58] D. D. Sleator and R. E. Tarjan. "A Data Structure for Dynamic Trees." In: *J. Comput. Syst. Sci.* 26.3 (1983), pp. 362–391.

[59] D. A. Spielman and S.-H. Teng. "A Local Clustering Algorithm for Massive Graphs and Its Application to Nearly Linear Time Graph Partitioning". In: *SIAM Journal on Computing* 42.1 (2013), pp. 1–26.

[60] D. A. Spielman and S.-H. Teng. "Nearly Linear Time Algorithms for Preconditioning and Solving Symmetric, Diagonally Dominant Linear Systems". In: *SIAM Journal on Matrix Analysis and Applications* 35.3 (2014), pp. 835–885.

[61] D. A. Spielman and S.-H. Teng. "Spectral Sparsification of Graphs". In: *SIAM J. Comput.* 40.4 (2011), pp. 981–1025.

[62] M. Unger, T. Pock, and H. Bischof. "Global Relabeling for Continuous Optimization in Binary Image Segmentation." In: *EMMCVPR*. Ed. by Y. Boykov et al. Vol. 6819. Lecture Notes in Computer Science. Springer, 2011, pp. 104–117.

[63] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course (Applied Optimization)*. Springer, 2004.

A. Appendix

A.1. Multiplicative weights update method

In this section a short overview over the multiplicative weights update method (MWU) is given. According to Christiano et al. their algorithm uses the MWU framework, but it is more a 'is inspired by'. The basic idea was used for some time and in different contexts (e.g. ADA Boost, ...) but the method was first generalized by Arora et al. [1].

A.1.1. The general framework

The every abstract setting for the MWU Framework is the following: There are random event and there are experts who can make predictions about the outcomes of these events. The quality of the experts differs, meaning that some experts make more accurate predictions than others. In order to achieve good prediction accuracy one is interested in identifying experts with high prediction accuracy.

Therefore, a weight is assigned to each expert, this weight corresponds to the (estimated) quality of the expert, i.e. experts with high weights are supposed to make more accurate predictions. Furthermore it is assumed that at the beginning no information about the quality of the experts is known, therefore, the weights are the same for each expert.

If a prediction about the upcoming event is needed, based on the weights an expert is chosen to make this prediction.

After the event happened and the outcome is known the weights of the experts are adjusted. If a expert would have made a wrong prediction then its weight is decreased and if the prediction would have been correct the weight is increased. If this is iterated for multiple events and their outcomes one can identify the reliable experts.

In order to formalize this idea let us define the following variables and functions. The number of experts is denoted by n and the number of all possible outcomes is denoted by m. The Matrix $\mathcal{M} \in \mathbb{R}^{n \times m}$ is the so called *penalty matrix* or *penalty function* where, $\mathcal{M}(i,j)$ denotes the penalty for expert i if the output of the event is j. Furthermore, it is assumed the values of $\mathcal{M}(i,j)$ are bounded by the width ρ:

$$\mathcal{M}(i,j) \in [-l, \rho], \qquad 0 < l \leq \rho, \quad \forall i,j$$

Note that the penalty can be positive as well, in this case the term 'reward' would be more adequate.

The variable t is used to indicate the number of the iteration. The weights, assigned to the experts after t iterations, are denoted by $w^t := (w_1^t, w_2^t, \ldots, w_n^t)$. As mentioned

above a random expert is chosen to make the prediction about the outcome of event t. The probability distribution is given by $p^t := {}^{w^t}/_{\sum_{j=1}^n w_j^t}$, therefore expert i is chosen with probability $p_i^t = {}^{w_i^t}/_{\sum_{j=1}^n w_j^t}$

All the distributions grouped together are denoted by $\mathcal{D}^t := (p_1^t, \ldots, p_n^t)$.

Algorithm 14 Multiplicative Weights Update method

1: **while** $t > 0$ **do**
2: pick expert according to $\mathcal{D}^t$ to predict outcome of the next event
3: let j^t denote the acutal outcome ▷ the real outcome
4: **for all** $i \in \{1, \ldots, n\}$ **do** ▷ update all weights
5: **if** $M(i, j^t) \geq 0$ **then**
6: $w_i^{t+1} \leftarrow w_i^t(1-\varepsilon)^{M(i,j^t)/\rho}$ ▷ punish
7: **else**
8: $w_i^{t+1} \leftarrow w_i^t(1+\varepsilon)^{-M(i,j^t)/\rho}$ ▷ reward

The multiplicative updates in line 6 and line 8 explain the name of the algorithm. In order to have a measurement of the quality of the prediction at iteration t one can define the s.c. 'expected penalty for outcome j^t'. This is denoted by $\mathcal{M}(\mathcal{D}^t, j^t)$ and simple is defined as:

$$\mathcal{M}(D^t, j^t) := \sum_{i=1}^{n} p_i^t \mathcal{M}(i, j^t)$$

Analog the 'expected total loss' after T rounds is given by $\sum_{t=1}^T \mathcal{M}(\mathcal{D}^t, j^t)$

Theorem 18 (Main theorem of MWU, Arora et al. [1]). *For given $\varepsilon \leq 1/2$ and for any expert i. After T rounds the expected total loss can be bounded by:*

$$\sum_{t=1}^{T} \mathcal{M}(\mathcal{D}^t, j^t) \leq \frac{\rho \ln n}{\varepsilon} + (1+\varepsilon) \sum_{\substack{1 \leq t \leq T \\ \mathcal{M}(i,j^t) \geq 0}} \mathcal{M}(i, j^t) + (1-\varepsilon) \sum_{\substack{1 \leq t \leq T \\ \mathcal{M}(i,j^t) < 0}} \mathcal{M}(i, j^t)$$

Remark 1. Theorem 18 also holds if the update step in algorithm 14 is formulated as

$$w_i^{t+1} = w_i^t(1 - \varepsilon \mathcal{M}(i, j^t))$$

In the following section the following corollary of theorem 18 will be used:

Corollary 6 (Arora et al. [1]). *Let $\delta > 0$ and $\varepsilon = \min(\frac{\delta}{4\rho}, \frac{1}{2})$. After $T = \frac{16\rho^2 \ln(n)}{\delta^2}$ iterations for every expert i the following bound holds:*

$$\frac{\sum_t \mathcal{M}(D^t, j^t)}{T} \leq \delta + \frac{\sum_t \mathcal{M}(i, j^t)}{T}$$

For proofs of Theorem 18 and corollary 6 see [1].

A.1.2. Solving Linear Programs via MWU Framework

One application of the MWU is to approximately solve linear programs, this method goes back to Plotkin et al. [49]. The algorithm of Christiano uses a similar results but with quite different proofs. The following method is given for completeness and may provide some structural insights.
The general question is if there exists a $x \in \mathbb{R}^n$, such that

$$Ax \geq b \qquad x \in P$$

where $A \in \mathbb{R}^{m \times n}$ and P is a convex set.
Furthermore, we assume that there exists an oracle which answers the question if a $x \in P$ exists, such that

$$c^t x \geq d \tag{A.1}$$

where $c = \sum_{j=1}^{m} p_j A_j$ (A_j is row j of A) and $d = p_i b_i$ for a probability distributions $(p_1, \ldots, p_m)$ and $i \in \{1, \ldots, m\}$. Therefore, one can rewrite (A.1):

$$\sum_{j=1}^{m} p_j A_j^t x = \sum_{j=1}^{m} p_j (Ax)_j \geq p_i b_i \tag{A.2}$$

Since one is interested in an approximation the goal is to compute a solution $x \in P$, with $A_i x \geq b_i - \delta$ or conclude that no such x exists, for some small δ.

The overall task is still to find a x satisfying all the constraints, whereas the oracle only has to find a x satisfying one constraint, which in general is easier to accomplish. This oracle will be queried multiple times with different constraints.

In the MWU terminology of experts and events, the experts corresponds to the constraints and the events corresponds to x. A experts prediction is correct if the constraint is satisfied, therefore, the penalties for expert i and event x is $A_i x - b_i$, i.e. the amount of how much a constraint is violoated. Furthermore, the penalties have to be restricted to $[-\rho, \rho]$. This is done by requiring the oracle to yield solutions x that satisfies this property.

Now the MWU-algorithm is run for T iterations and in each iteration the oracle is called with $c = \sum_i p_i A_i$ and $d = \sum_i p_i b_i$, if the oracle fails at some iteration, the return value is `fail`. If all oracle calls returned a x^t then Corollary 6 with $\varepsilon = \frac{\delta}{4\rho}$ yields,

$$\underbrace{\frac{\sum_{t=1}^{T} \sum_j (A_j x^t - b_j)}{T}}_{\geq 0} \leq \delta + \frac{\sum_{t=1}^{T} (A_i x^t - b_i)}{T}$$

which implies that $A_i \sum_{i=1}^{T} {}^{x^t}/_T \geq b_i - \delta$ and ,therefore, $\bar{x} := \sum_{i=1}^{T} {}^{x^t}/_T$ is the desired approximate solution. On the other hand, if the oracle does not find a solution then one can assume that the system is infeasible. For a detailed proof see the paper of Arora et al..

A closer look reveals that if the penalty for expert i is greater than zero, this means the constraint is fulfilled, the weight is decreased. On the other hand if the constraint is violated the weight gets increased. Therefore, violated constraints are getting more critical in (A.2).

A.2. Nesterov's Accelerated Gradient Descent Method

This section gives a short overview of Nesterov's Accelerated Gradient Descent Method. This is a accelerated Gradient descent method, i.e. it takes modified gradient steps and converges faster.

Basically, minimization problems of the following form are considered

$$\min_{x \in \mathbb{R}^n} f(x) \tag{P1}$$

where f is smooth, convex, differentiable and the gradient has Lipschitz constant L. The standard method for (iteratively) solving problems of this form is 'gradient descent'. The main idea of gradient descent is that (in some neighbourhood of x) the gradient of f at a point x points into the direction of the strongest ascent. Therefore, a (small) step in the opposite direction of the gradient would yield the strongest descent of the function.

Algorithm 15 Gradient descent

choose x_0 and a stepsize α
while not converge **do**
$x_{i+1} = x_i - \alpha \nabla f(x_i)$

It is well known that this algorithm converges for $\alpha \in (0, 2/L)$ with convergence rate $O(1/k)$, see for instance[63].

In Nesterov's Accelerated Gradient Descent Method the steps of Algorithm 15 are modified and a momentum step is added.

Algorithm 16 Nesterov's Gradient method

function NESTEROV($\nabla f, L, c, T, x_0$)
set $i = 0$, $z_0 = x_0$ and choose a stepsize α
while $i < T$ **do**
$x_{i+1} = z_i - 1/L \nabla f(z_i)$
if c $= 0$ **then**
$\alpha_{k+1} = \frac{(1+\sqrt{4\alpha_k^2+2})}{2}$
$z_{k+1} = x_k + \frac{\alpha_k - 1}{\alpha_{k+1}}(x_k - x_{k-1})$
else
$z_{k+1} = x_k + \frac{\sqrt{L}-\sqrt{c}}{\sqrt{L}+\sqrt{c}}(x_k - x_{k-1})$
return x_T

The parameter c denotes the strong convexity parameter of f. In general this algorithm has a convergence rate of $O(1/k^2)$ which already is a huge improvement over the simple gradient descent method. The algorithm converges even faster if f is strongly convex with parameter $c > 0$. In this case after $O(\log 1/\varepsilon)$ iterations the absolute error is less than ε. Furthermore, if only a ε approximation is required one can bound the number of necessary iterations:

Theorem 19 (Nesterov). *Let $f : \mathbb{R}^n \to \mathbb{R}$, L the Lipschitz parameter of ∇F and $c > 0$ the strong convexity parameter. Let x^* be the optimal solution of (P1) and x_0 some initial vector. Algorithm 16 produces a vector x_T with*

$$f(x_T) - f(x^*) \leq \varepsilon$$

after

$$T = \min\left(2\sqrt{\frac{L}{\varepsilon}}\|x_0 - x^*\|_2, \sqrt{\frac{L}{c}}\log\left(\frac{4\|x_0 - x^*\|_2}{\varepsilon}\right)\right)$$

iterations.